TDP Documentação 3.0.1

Português - Brasil

TECNISYS

© Copyright Tecnisys 2026. All rights reserved.

Sumário

PARTE I - INTRODUÇÃO

Visão Geral

Início Rápido

Início Rápido

TDP Datacenter

TDP Kubernetes

Conceitos

Kubernetes

Apache Airflow

Apache Ambari

Apache Atlas

ClickHouse

CloudBeaver

Delta Lake

Apache Druid

Apache Hadoop

Apache HDFS

MapReduce

Apache Hive

Apache Flink

Great Expectations

Apache HBase

Apache Iceberg

JupyterLab

Apache Kafka

Kerberos

Apache Knox

Apache Livy

© Copyright Tecnisys 2026. All rights reserved.

Apache NiFi
OpenMetadata
Apache Ozone
Apache Ranger
Apache Ranger KMS
Apache Spark
Apache Sqoop
Apache Solr
Apache Superset
Apache Tez
Trino
Apache YARN
Apache Zeppelin
Apache ZooKeeper

PARTE II - TDP DATACENTER

Instalação

Requisitos Mínimos
Ferramentas de Apoio
Preparação do Ambiente
Pacotes de Instalação
Instalação do Apache Ambari
Criação do Cluster e Instalação dos Componentes
Instalação do tdpctl

Atualização

Fluxograma de Atualização
Pré-Requisitos para Atualização
Atualização do Apache Ambari
Registro da Nova Versão
Atualização dos Componentes

PARTE III - TDP KUBERNETES

© Copyright Tecnisys 2026. All rights reserved.

Instalação

- Pré-Requisitos
- Instalação via Helm
- Argo CD
- Políticas de Cluster
- Registry Privado com Kyverno

Configuração

- Configuração Geral
- Exposição externa
- Segurança — Padrões Gerais
- Integrações — Padrões Gerais
- Airflow
- Airflow – Segurança
- Airflow – Integração
- Argo CD
- Argo CD – Segurança
- ClickHouse
- ClickHouse – Segurança
- ClickHouse – Integração
- CloudBeaver
- CloudBeaver – Segurança
- CloudBeaver – Integração
- Delta Lake
- Delta Lake – Segurança
- Delta Lake – Integração
- Hive Metastore
- Hive Metastore – Segurança
- Hive Metastore – Integração
- Iceberg
- Iceberg – Segurança
- Iceberg – Integração
- JupyterLab
- JupyterLab – Segurança

© Copyright Tecnisys 2026. All rights reserved.

JupyterLab – Integração
Kafka
Kafka – Segurança
Kafka – Integração
NiFi
NiFi – Segurança
NiFi – Integração
OpenMetadata
OpenMetadata – Segurança
OpenMetadata – Integração
Apache Ozone
Apache Ozone – Segurança
Apache Ozone – Integração
PostgreSQL
PostgreSQL – Integração
Ranger
Ranger – Segurança
Ranger – Integração
Spark
Spark – Segurança
Spark – Integração
Superset
Superset – Segurança
Superset – Integração
Trino
Trino – Segurança
Trino – Integração

Atualização

Atualização via Helm
Atualização via ArgoCD
Rollback

Desinstalação

Desinstalação

© Copyright Tecnisys 2026. All rights reserved.



PARTE IV - EXTENSÕES

Extensões

Extensão TDP APIM Insights

Instalação do add-on para o WSO2

PARTE V - COMPONENTES

Matriz de Versões

Componentes TDP Datacenter

Componentes TDP Kubernetes

PARTE VI - NOTAS TÉCNICAS

Notas Técnicas

Componentes Adicionados

Componentes Removidos

Destaques

Destaques – TDP Datacenter

Destaques – TDP Kubernetes

Mais Informações

Mais Informações

Mais Informações – TDP Datacenter

Mais Informações – TDP Kubernetes

Notas de Versão

TDP Datacenter – Notas

TDP Datacenter 3.0.0

TDP Kubernetes – Notas

TDP Kubernetes 3.0.0

TDP Kubernetes 3.0.1

Correções

Correções – TDP Datacenter

Correções – TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.



PARTE VII - ROADMAP

Roadmap

PARTE VIII - OUTRAS INFORMAÇÕES

Suporte

Informações Legais

© Copyright Tecnisys 2026. All rights reserved.

PARTE I - INTRODUÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Visão Geral

Descubra o que é o TDP e como ele pode ajudar sua organização a desenvolver modelos de negócio orientados a dados.

O que é o TDP?

A Tecnisys Data Platform (TDP) é a Plataforma de Dados da Tecnisys para análise e governança de dados. Nosso objetivo é auxiliar organizações no desenvolvimento de modelos de negócio orientados a dados.

O TDP é formado somente por projetos de código aberto, não requer licença de uso ou subscrição ativa para operar plenamente, e oferece os principais componentes open source de uma Stack de Dados Moderna, tais como Apache Spark, Apache Iceberg, Trino, Apache Kafka, Apache Flink, Apache Druid, Apache NiFi, Apache Airflow, Apache Superset e tantos outros.



Figura 1 - Principais componentes

© Copyright Tecnisys 2026. All rights reserved.

Com o TDP, é possível ir da ingestão à análise de dados, criando pipelines integrados e fluxos de dados contínuos para diferentes tipos de workload, seja streaming ou batch. A Plataforma permite explorar e analisar rapidamente grandes volumes de dados, treinar modelos de aprendizagem de máquina, apresentar gráficos dinâmicos e gerar conhecimento estratégico para a tomada de decisões. Tudo isso com segurança, auditoria e governança de dados de ponta a ponta.

Acesse o site da Tecnisys e inscreva-se para baixar gratuitamente essa incrível Plataforma de Dados.

Por que usar o TDP?

O TDP possibilita a implantação de um Ambiente de Dados completo e integrado em apenas alguns passos guiados. Além disso, a Plataforma conta com uma interface gráfica completa para a administração centralizada de todas as máquinas e serviços do Cluster.

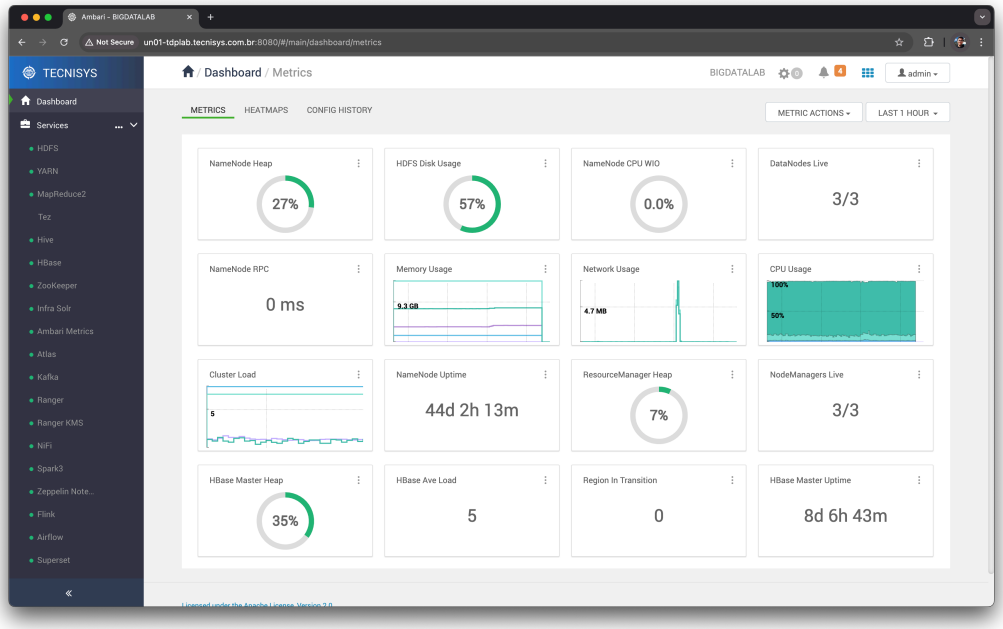


Figura 2 - Interface gráfica para a administração de máquinas e serviços

Entre as soluções disponibilizadas pela Plataforma TDP, destacam-se:

- Administração centralizada
- Armazenamento e processamento distribuído
- Análise de dados e aprendizagem de máquina descentralizados
- Virtualização de dados

© Copyright Tecnisys 2026. All rights reserved.



- Pipelines de dados automatizados
- Fluxos de dados flexíveis
- Desenvolvimento colaborativo
- Busca e indexação textual
- Barramento de dados
- Segurança avançada
- Governança de dados
- Visualização de dados dinâmica

O TDP é uma Plataforma de Dados Brasileira em constante evolução, desenvolvida por um time de engenheiros de software e especialistas atentos às necessidades organizacionais.

© Copyright Tecnisys 2026. All rights reserved.

INÍCIO RÁPIDO

© Copyright Tecnisys 2026. All rights reserved.

Início Rápido

Descubra como iniciar rapidamente com o TDP através da implantação em máquina virtual ou cluster Kubernetes.

- **TDP Datacenter**: Guia para importação e execução do TDP Datacenter em máquina virtual.
- **TDP Kubernetes**: Guia para implantação do TDP Kubernetes em cluster Kubernetes.

© Copyright Tecnisys 2026. All rights reserved.

TDP Datacenter

> **Pré-Requisitos**
>
> - Sistema Operacional: Windows 10 ou superior, ou Linux com VirtualBox 7 ou superior
> - Processador: 4 vCPUs disponíveis
> - Memória RAM: 10 GB disponíveis
> - Espaço disponível em disco: Inicialmente, 100 GB, podendo chegar a 140 GB
> - VirtualBox instalado e configurado

O que é a Sandbox do TDP Datacenter?

A Sandbox do TDP Datacenter é uma máquina virtual pré-configurada que emula um cluster TDP Datacenter com apenas um nó. Ela permite explorar, de forma rápida e simplificada — principalmente para estudo ou testes — funcionalidades como:

- **Ingestão e transformação de dados**.
- **Execução de pipeline de dados**.
- **Testes de integrações com ferramentas analíticas**.

Esta solução é ideal para desenvolvedores, analistas de dados e administradores que desejam experimentar ou testar algumas das funcionalidades avançadas de transformação, processamento e análise de dados antes de implementá-las em produção, sem a complexidade de uma instalação completa.

Este guia o auxiliará na importação, configuração, acesso e uso da Sandbox do TDP Datacenter.

Importação da Sandbox

Siga as instruções abaixo para importar e iniciar a Sandbox:

© Copyright Tecnisys 2026. All rights reserved.

1. Faça o download da Sandbox (arquivo `.ova`), utilizando o Repositório Público de Pacotes da Tecnisys.

   No Repositório Público de Pacotes da Tecnisys, os arquivos da Sandbox estão disponíveis no diretório `/sandbox/tdp/`.

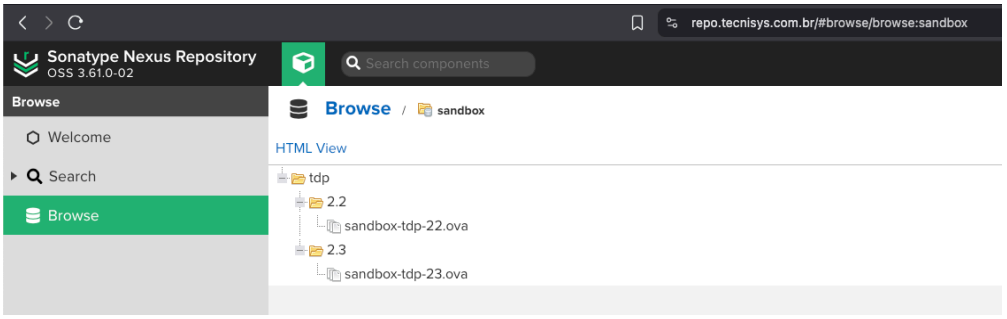


Figura 1 - Repositório Público de Pacotes da Tecnisys

> **NOTE**
>
> os arquivos .ova seguem o padrão => sandbox-tdp-xxx-nnn-ova, sendo:
>
> - xxx - Versão do TDP (exemplo: 230)
> - nnn - numero sequencial (ex: 001)

2. Importe a imagem da Sandbox:

   2.1. Abra o VirtualBox.
   2.2. No menu superior, clique em Arquivo → Importar Appliance.

© Copyright Tecnisys 2026. All rights reserved.

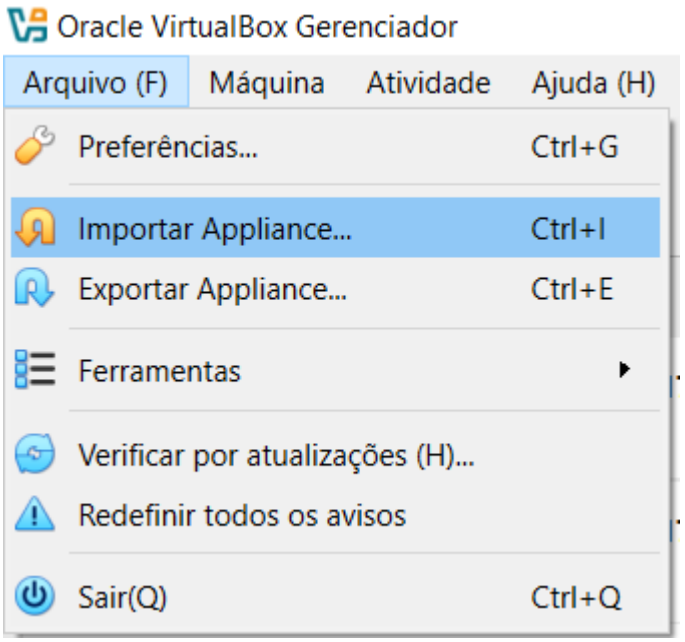


Figura 2 - Importação do arquivo .ova

2.3. Na janela que se abre, selecione o arquivo `.ova` da Sandbox que você baixou.
2.4. Clique em `Finalizar` para iniciar a importação da imagem da máquina virtual (VM).

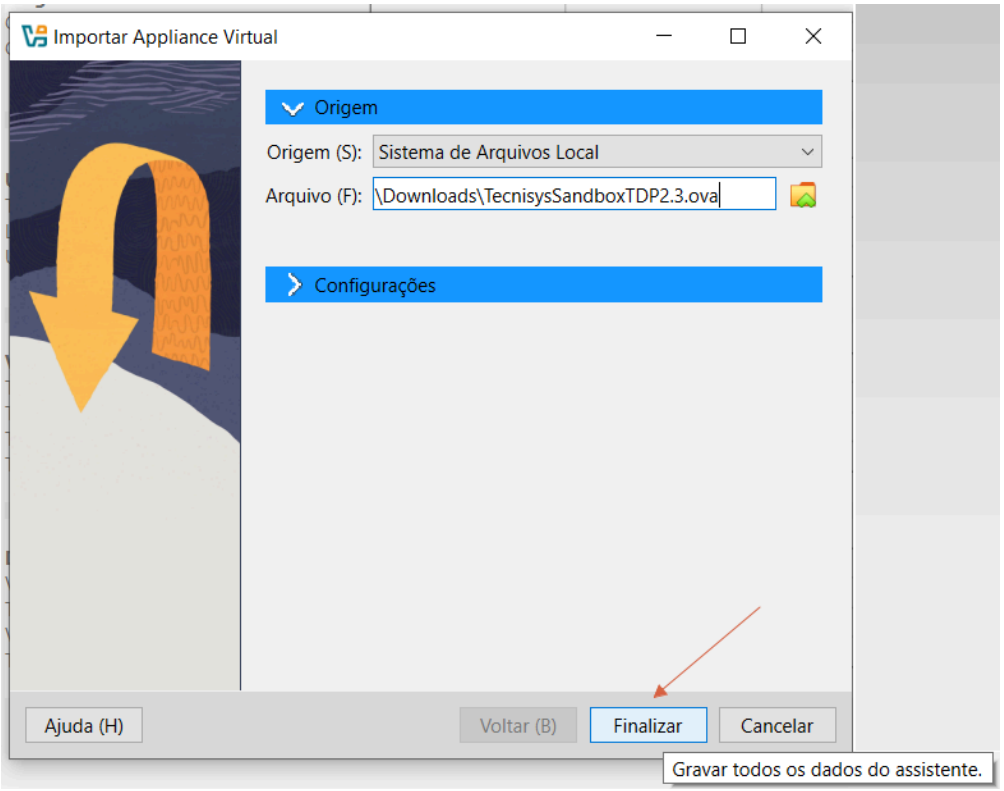


Figura 3 - Importação do arquivo .ova

© Copyright Tecnisys 2026. All rights reserved.

2.5. Aceite a licença de uso, clicando em `Concordo`.

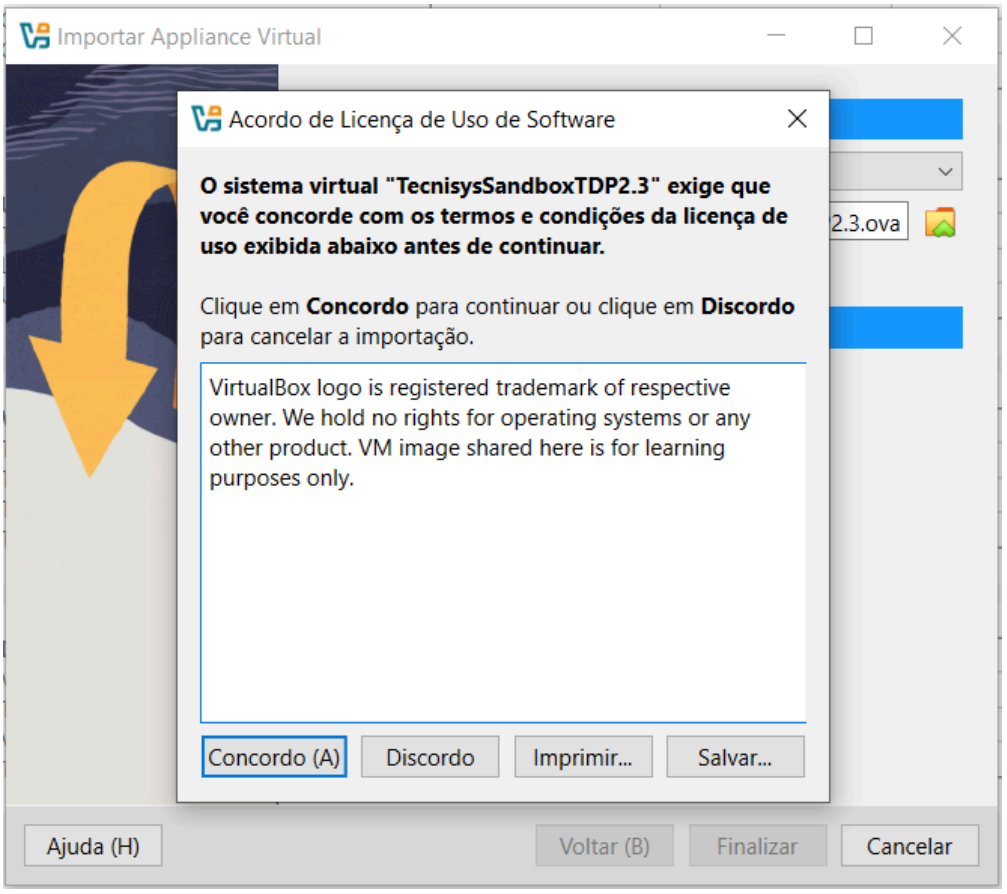


Figura 4 - Aceite dos termos e condições da licença de uso

2.6. Aguarde a importação da imagem da VM ser concluída.

2.7. Inicie a VM importada.

© Copyright Tecnisys 2026. All rights reserved.

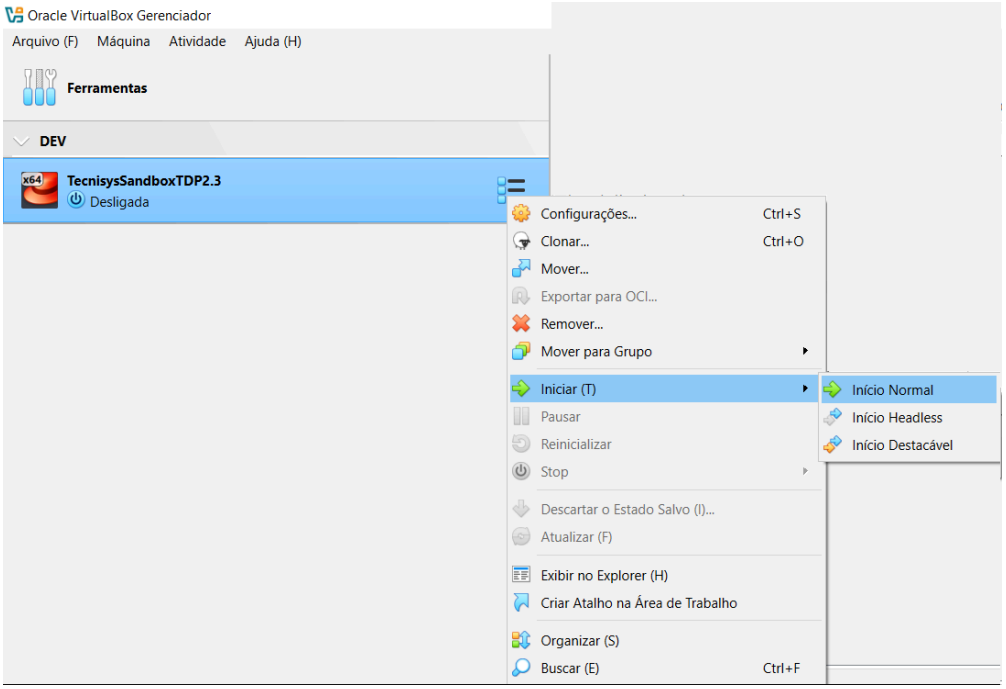


Figura 5 - Inicialização da Máquina Virtual

2.8. Aguarde a VM iniciar e exibir a tela inicial customizada.

2.9. A Sandbox está pronta para uso!

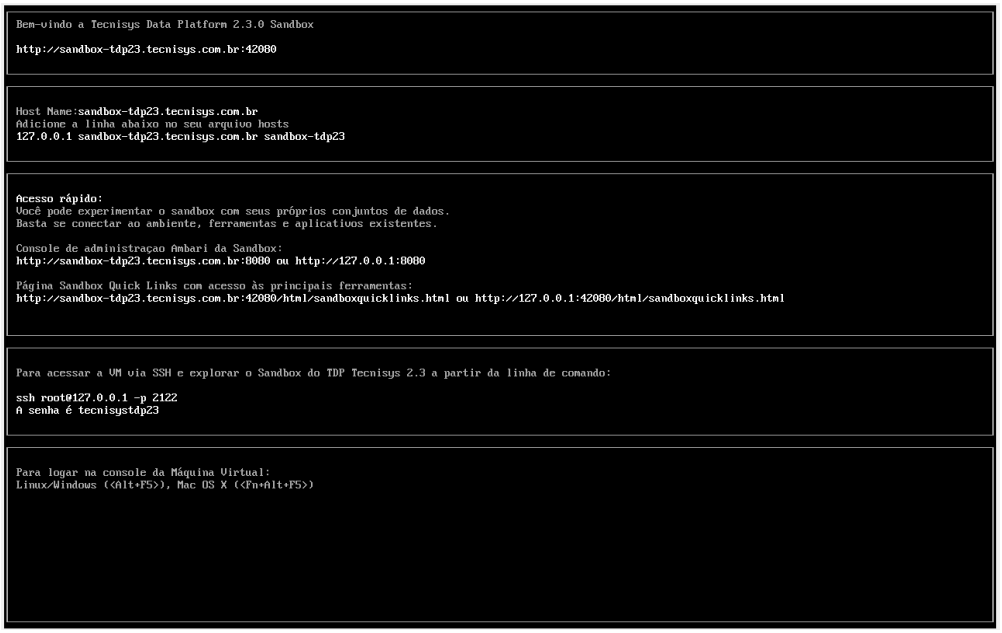


Figura 6 - SandBox pronta para uso

3. Configure a resolução de nomes da Sandbox:

Ao utilizar uma VM local para testes e estudos, é recomendável mapear o domínio da VM localmente. Essa configuração facilita o acesso ao ambiente, permite o

© Copyright Tecnisys 2026. All rights reserved.

redirecionamento de portas e simula de forma mais realista um ambiente de produção, eliminando a dependência de servidores DNS externos.

3.1. Configure o arquivo de `hosts` no host local (Windows ou Linux), associando o IP 127.0.0.1 ao hostname (FQDN) da sandbox (`sandbox-tdp<MAJOR VERSION><MINOR VERSION>.tecnisys.com.br`). Por exemplo: `sandbox-tdp23.tecnisys.com.br`.

> **NOTE**
>
> O arquivo de `hosts` é um arquivo de configuração local presente em sistemas operacionais como Windows (`C:/Windows/System32/drivers/etc/hosts`) e Linux (`/etc/hosts`). Ele serve para resolver nomes de domínio para IPs, eliminando a necessidade de consultar um servidor de DNS externo.

3.1.a. Configuração do arquivo de hosts no **Linux**.

3.1.a.1. Abra o arquivo de `hosts` com um editor de texto de sua escolha. Neste tutorial, usamos o `vi`:

Terminal input

```
sudo vi /etc/hosts
```

3.1.a.2. Adicione ao final do arquivo uma nova linha (no `vi`, tecle `G` para ir ao final do arquivo e `i` para inserir uma nova linha) com a resolução do IP `127.0.0.1` para o hostname da sandbox. Note que após o IP deve ser informado o hostname com o domínio (FQDN), seguido de um espaço, e o hostname apenas. Veja o exemplo abaixo para a Sandbox do TDP 2.3:

Terminal input

```
127.0.0.1 sandbox-tdp23.tecnisys.com.br sandbox-tdp23
```

3.1.a.3. Salve e feche o arquivo (no `vi`, tecle `ESC` e digite `:wq`).

© Copyright Tecnisys 2026. All rights reserved.

3.1.b. Configuração do arquivo de hosts no **Windows**.

3.1.b.1. Abra o Bloco de Notas como Administrador.

> **TIP**
>
> Clique com o botão direito no ícone do Bloco de Notas e selecione `Executar como Administrador`.

3.1.b.2. No Bloco de Notas, abra o arquivo `C:/Windows/System32/drivers/etc/hosts`.

3.1.b.3. Adicione ao final do arquivo uma nova linha com a resolução do IP `127.0.0.1` para o hostname da sandbox. Note que após o IP deve ser informado o hostname com o domínio (FQDN), seguido de um espaço, e o hostname apenas. Veja o exemplo abaixo para a Sandbox do TDP 2.3:

Terminal input

```
127.0.0.1 sandbox-tdp23.tecnisys.com.br sandbox-tdp23
```

3.1.b.4. Salve o arquivo.

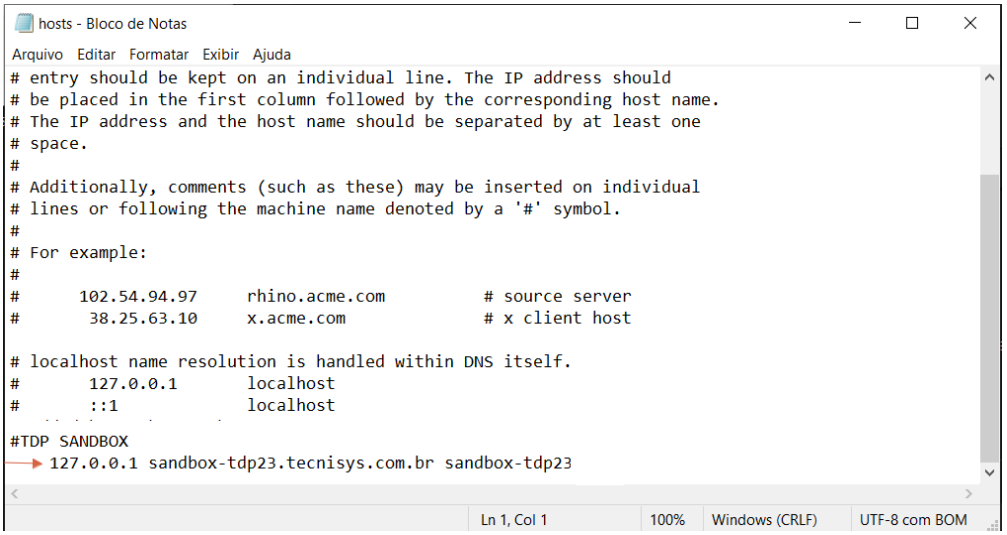


Figura 7 - Alteração do arquivo de hosts no Windows

Acesso ao Cluster TDP

© Copyright Tecnisys 2026. All rights reserved.

Uma vez que a VM esteja em execução, utilize o navegador ou o terminal para acessar e interagir com os serviços do cluster TDP na Sandbox.

Página de Início da Sandbox

1. Abra o seu navegador e acesse a página de Início da Sandbox através da seguinte URL:

   ```
   http://localhost:42080
   ```

   Caso tenha sido adicionado localmente uma resolução de nomes para a VM da Sandbox, utilize o hostname definido no arquivo de `hosts`. Por exemplo, para a Sandbox do TDP 2.3:

   ```
   http://sandbox-tdp23.tecnisys.com.br:42080
   ```

2. Na página de Início da Sandbox temos a área de `Quick Links` para acesso rápido às interfaces web de componentes do Cluster TDP. Selecione o card do componente desejado para visualizar a URL de acesso e as credenciais de acesso.



Figura 8 - Página de Início da Sandbox

Inicialização de Serviços do Cluster TDP

Através da página de Início da Sandbox, acesse a interface web do Apache Ambari, o componente de administração centralizada da plataforma TDP, para iniciar os serviços do Cluster TDP, como Apache Kafka, Apache NiFi, Trino, entre outros.

© Copyright Tecnisys 2026. All rights reserved.

> **WARNING**
>
> Evite iniciar muitos serviços ao mesmo tempo, pois isso pode levar a problemas de desempenho e instabilidade em uma VM local com recursos computacionais limitados.

1. Na tela de login do Apache Ambari, insira as credenciais abaixo para realizar o acesso:
    - **Username**: admin
    - **Password**: admin
2. Selecione o serviço de interesse no menu lateral esquerdo.
3. Inicie todos os componentes do serviço clicando no botão `Actions` → `Restart All` localizado no lado superior direito.

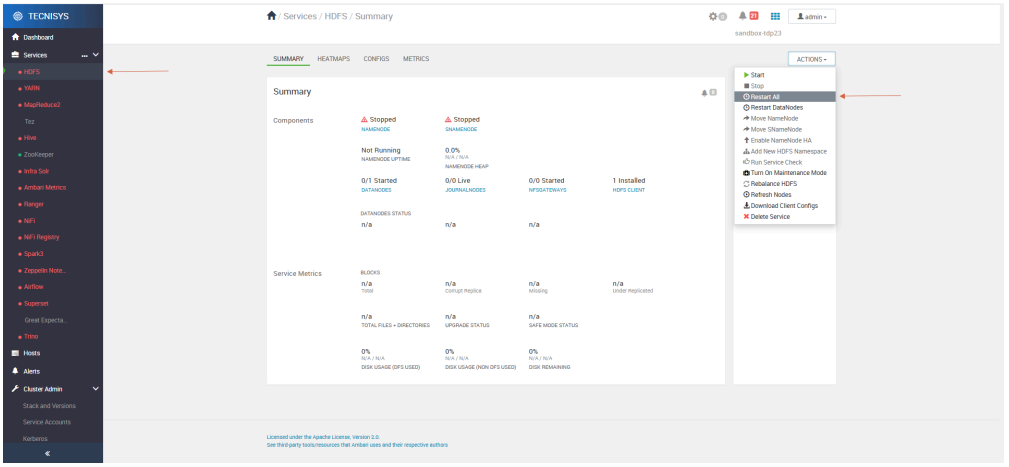


Figura 9 - Iniciar serviço

Acesso à Sandbox via SSH

Através de um terminal, é possível acessar a Sandbox realizando uma conexão SSH na porta 2122 da VM:

Terminal input

```
ssh -p 2122 devopstdp@localhost
```

Credenciais de Acesso

© Copyright Tecnisys 2026. All rights reserved.



Seguem as credenciais de acesso dos usuários previamente criados na VM da Sandbox:

TDP 2.3

Usuário	Senha
root	tecnisystdp23
sandbox	tecnisystdp23
devopstdp	tecnisystdp23

TDP 2.2

Usuário	Senha
root	tecnisystdp22
sandbox	tecnisystdp22
devopstdp	tecnisystdp22

A seguir, são listadas as principais credenciais dos usuários de serviços do Cluster TDP na Sandbox:

TDP 2.3

Serviço	Usuário	Senha
Airflow	admin	tecnisystdp23
Ambari	admin	admin
Druid Database User	druid	druid
Grafana	admin	tecnisystdp23

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Usuário	Senha
Hive Database User	hive	hive
Kafka UI	kafka-ui	tecnisystdp23
Ranger	admin	tecnisystdp23
Ranger Admin Credentials for Ambari	amb_ranger_admin	tecnisystdp23
Ranger Database User	ranger	ranger

TDP 2.2

Serviço	Usuário	Senha
Airflow	admin	tecnisystdp22
Ambari	admin	admin
Druid Database User	druid	druid
Grafana	admin	tecnisystdp22
Hive Database User	hive	hive
Kafka UI	kafka-ui	tecnisystdp22
Ranger	admin	tecnisystdp22
Ranger Admin Credentials for Ambari	amb_ranger_admin	tecnisystdp22
Ranger Database User	ranger	ranger

© Copyright Tecnisys 2026. All rights reserved.



Por último, seguem os valores de propriedades sensíveis do Cluster TDP na Sandbox:

TDP 2.3

Serviço	Propriedade	Valor
Knox	Master Secret	tecnisystdp23
NiFi	CA Token	tecnisystdp23
NiFi	Encryption Password	tecnisystdp23
NiFi	Registry Encryption Password	tecnisystdp23
NiFi	Sensitive property values encryption password	tecnisystdp23
Ranger	Usersync User's Password	tecnisystdp23
Ranger	Tagsync User's Password	tecnisystdp23
Superset	SECRET_KEY	tecnisystdp23
Superset	Superset Admin Email	admin@tecnisys.com.br
Superset	Superset Admin Firstname	admin
Superset	Superset Admin Lastname	tecnisys
Superset	Superset Admin Password	tecnisystdp23

TDP 2.2

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Propriedade	Valor
Knox	Master Secret	tecnisystdp22
NiFi	CA Token	tecnisystdp22
NiFi	Encryption Password	tecnisystdp22
NiFi	Registry Encryption Password	tecnisystdp22
NiFi	Sensitive property values encryption password	tecnisystdp22
Ranger	Usersync User's Password	tecnisystdp22
Ranger	Tagsync User's Password	tecnisystdp22
Superset	SECRET_KEY	tecnisystdp22
Superset	Superset Admin Email	sandbox-tdp22@tecnisys.com.br
Superset	Superset Admin Firstname	tecnisys
Superset	Superset Admin Lastname	tdp22
Superset	Superset Admin Password	tecnisystdp22

Portas de Comunicação

A tabela abaixo lista os serviços e as respectivas portas de comunicação utilizadas na Sandbox. Esta configuração permite o acesso aos serviços hospedados na máquina

© Copyright Tecnisys 2026. All rights reserved.

virtual.

Aplicação	Porta
Ambari	8080
Apache	42080
Datanode	50075
Atlas	21000
Grafana	3000
Druid Router	8888
Druid Coordinator	8081
HBaseMaster	16010
HBaseRegion	16030
HDFS	8020
HiveJDBCJar	10002
HostSSH	2122
HostSSH	2200
JobHistory	19888
Kafka	8079
Knox	8443

© Copyright Tecnisys 2026. All rights reserved.

Aplicação	Porta
NiFi	9090
NiFi Registry	61080
NodeManager	8042
Nodemanager	8040
ResourceManager	8032
Infra Solr	8886
Solr Admin	8983
SparkHistoryServer	18082
WebHBase	60080
Ranger	50070
Ranger	6080
YARN	8088
YarnATS	8188
Zeppelin	9995
Superset	9088
AirflowUI	18080
CeleryFlowerUI	5555

© Copyright Tecnisys 2026. All rights reserved.

Aplicação	Porta
Trino	5555

Utilize essa tabela para acessar e configurar os serviços conforme necessário.

> **WARNING**
>
> Caso a sua máquina local já esteja utilizando alguma das portas configuradas para a Sandbox, a VM não será inicializada, e o VirtualBox apresentará um erro durante o processo, abortando a execução.

© Copyright Tecnisys 2026. All rights reserved.

TDP Kubernetes

Pré-Requisitos

- Cluster Kubernetes 1.32+
- kubectl instalado e configurado
- Helm 3.2.0+
- Acesso ao registry OCI da Tecnisys (registry.tecnisys.com.br)
- Recursos mínimos recomendados no cluster:
  - CPU: 16 vCPUs disponíveis
  - Memória RAM: 32 GB disponíveis
  - Armazenamento persistente: 200 GB disponíveis

O que é o TDP Kubernetes?

O TDP Kubernetes é a edição cloud-native da plataforma TDP, projetada para execução em clusters Kubernetes. Esta edição utiliza contêineres e orquestração para fornecer:

- **Escalabilidade horizontal automática** de componentes.
- **Alta disponibilidade** com failover automático.
- **Implantação declarativa** via Helm Charts ou ArgoCD.
- **Isolamento de recursos** e gerenciamento eficiente.

Esta solução é ideal para equipes que buscam uma plataforma de dados moderna, escalável e baseada em práticas DevOps e GitOps.

Este guia o auxiliará na implantação, configuração e acesso ao TDP Kubernetes em um cluster Kubernetes.

1. Preparação do Ambiente

Antes de iniciar a instalação, certifique-se de que o cluster Kubernetes está configurado e acessível.

© Copyright Tecnisys 2026. All rights reserved.

1.1. Verificar acesso ao cluster

Execute o comando abaixo para validar o acesso ao cluster Kubernetes:

Terminal input

```
kubectl cluster-info
```

A saída deve exibir informações sobre o cluster, incluindo o endereço do API server.

1.2. Criar namespace para o TDP

Crie um namespace dedicado para os componentes do TDP. O nome é livre — escolha um identificador alinhado à sua política de namespaces (por exemplo `tdp-homologacao`, `data-platform`).

Terminal input

```
kubectl create namespace <NAMESPACE>
```

EXEMPLO

Nomes usados em trechos ilustrativos mais abaixo: `tdp-project` como namespace e releases alinhados ao nome do chart (por exemplo `tdp-kafka`).

Exemplo

```
kubectl create namespace tdp-project
```

1.3. Configurar acesso ao registry OCI

Os Helm Charts do TDP Kubernetes estão hospedados no registry OCI da Tecnisys. Configure o acesso com suas credenciais:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm registry login registry.tecnisys.com.br \
  --username <username> \
  --password <password>
```

> **NOTE**
>
> Substitua `<username>` e `<password>` pelas credenciais fornecidas pela Tecnisys.

2. Instalação via Helm Charts

O Helm é o método recomendado para instalação de componentes individuais do TDP Kubernetes.

2.1. Adicionar o repositório Helm (se aplicável)

Caso o registry utilize repositório Helm tradicional, adicione-o:

Terminal input

```
helm repo add tecnisys https://registry.tecnisys.com.br/helm
helm repo update
```

2.2. Instalar componentes do TDP

Cada componente do TDP possui seu próprio Helm Chart.

Instalação ilustrativa do Apache Kafka (substitua `<RELEASE_NAME>` pelo nome do release Helm e `<NAMESPACE>` pelo namespace alvo):

Terminal input

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka
\
  --namespace <NAMESPACE> \
  --version <CHART_VERSION> \
  --create-namespace \
  -f <VALUES_FILE>
```

© Copyright Tecnisys 2026. All rights reserved.

EXEMPLO

Exemplo

```
helm install tdp-kafka oci://registry.tecnisys.com.br/tdp/charts/tdp-
kafka \
  --namespace tdp-project \
  --version 3.0.1 \
  --create-namespace \
  -f values-kafka.yaml
```

2.3. Personalizar a instalação

Você pode personalizar a instalação criando um arquivo `values.yaml`:

values.yaml

```
kafka:
  replicas: 3
  resources:
    requests:
      memory: "4Gi"
      cpu: "2"
    limits:
      memory: "8Gi"
      cpu: "4"
  persistence:
    enabled: true
    size: 100Gi
```

Aplique a configuração durante a instalação:

Terminal input

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka
\
  --namespace <NAMESPACE> \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  --version <CHART_VERSION> \
  -f <VALUES_FILE>
```

EXEMPLO

Exemplo

```
helm install tdp-kafka oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka \
  --namespace tdp-project \
  --version 3.0.1 \
  -f values.yaml
```

2.4. Verificar a instalação

Liste os recursos criados no namespace:

Terminal input

```
kubectl get all -n <NAMESPACE>
```

Verifique o status dos Pods:

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

Todos os Pods devem estar com status `Running` após alguns minutos.

3. Instalação via ArgoCD (GitOps)

O ArgoCD permite gerenciar a implantação dos componentes de forma declarativa via Git.

3.1. Instalar o ArgoCD

Se o ArgoCD ainda não estiver instalado no cluster, instale-o com os comandos:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

3.2. Acessar a interface do ArgoCD

Exponha o serviço do ArgoCD:

Terminal input

```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

Acesse a interface em `https://localhost:8080`.

A senha inicial do usuário `admin` pode ser obtida com:

Terminal input

```
kubectl get secret argocd-initial-admin-secret -n argocd \
  -o jsonpath="{.data.password}" | base64 -d
```

3.3. Criar Application no ArgoCD

Crie um arquivo YAML descrevendo a aplicação TDP, conforme exemplo:

tdp-application.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: tdp-kafka
  namespace: argocd
spec:
  project: default
  source:
    repoURL: oci://registry.tecnisys.com.br/tdp/charts
```

© Copyright Tecnisys 2026. All rights reserved.

```
    chart: tdp-kafka
    targetRevision: 3.0.1
  destination:
    server: https://kubernetes.default.svc
    namespace: tdp-project
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

Aplique a configuração:

Terminal input

```
kubectl apply -f <application-manifest>
```

EXEMPLO

Exemplo

```
kubectl apply -f tdp-application.yaml
```

O ArgoCD sincronizará automaticamente o estado desejado com o cluster.

4. Acesso aos Componentes

Após a instalação, os componentes podem ser acessados via Services do Kubernetes.

4.1. Listar Services disponíveis

Terminal input

```
kubectl get services -n <NAMESPACE>
```

4.2. Acessar interfaces web via Port Forwarding

© Copyright Tecnisys 2026. All rights reserved.

Para expor a UI do Kafka localmente:

Terminal input

```
kubectl port-forward svc/kafka-ui 8080:80 -n <NAMESPACE>
```

> **EXEMPLO**
>
> Em instalações Strimzi comuns, o serviço da UI pode aparecer como `tdp-kafka-kafka-ui` (ajuste ao seu cluster).
>
> Exemplo
>
> ```
> kubectl port-forward svc/tdp-kafka-kafka-ui 8080:80 -n tdp-project
> ```

Acesse a interface em `http://localhost:8080`.

4.3. Configurar Ingress (opcional)

Para acesso externo permanente, configure um Ingress Controller (como NGINX Ingress).

Manifesto genérico (substitua `<NAMESPACE>`, `<INGRESS_NAME>` e `<KAFKA_UI_HOSTNAME>`):

ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <INGRESS_NAME>
  namespace: <NAMESPACE>
spec:
  rules:
  - host: <KAFKA_UI_HOSTNAME>
    http:
      paths:
      - path: /
        pathType: Prefix
```

© Copyright Tecnisys 2026. All rights reserved.

```
      backend:
        service:
          name: kafka-ui
          port:
            number: 80
```

EXEMPLO

Valores ilustrativos:

Exemplo

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kafka-ui-ingress
  namespace: tdp-project
spec:
  rules:
  - host: kafka-ui.tdp.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: tdp-kafka-ui
            port:
              number: 80
```

Aplique o Ingress:

Terminal input

```
kubectl apply -f <ingress-manifest>
```

EXEMPLO

© Copyright Tecnisys 2026. All rights reserved.

Exemplo

```
kubectl apply -f kafka-ui-ingress.yaml
```

Configure a resolução de DNS local (`/etc/hosts` no Linux ou `C:\Windows\System32\drivers\etc\hosts` no Windows):

```
<INGRESS_IP>   <hostname>
```

EXEMPLO

```
192.168.1.50   kafka-ui.tdp.local
```

5. Principais Componentes do TDP Kubernetes

O TDP Kubernetes é composto por diversos componentes organizados em camadas:

Componente	Descrição	Helm Chart
Apache Kafka	Streaming de eventos	`tdp-kafka`
Apache NiFi	Orquestração de fluxos de dados	`tdp-nifi`
Trino	Motor de consultas SQL distribuídas	`tdp-trino`
Apache Airflow	Orquestração de workflows	`tdp-airflow`
Apache Superset	Visualização e BI	`tdp-superset`

© Copyright Tecnisys 2026. All rights reserved.

Componente	Descrição	Helm Chart
PostgreSQL	Banco de dados relacional	`tdp-postgresql`
ClickHouse	Banco de dados OLAP	`tdp-clickhouse`

Para instalar múltiplos componentes, repita os passos de instalação via Helm ou ArgoCD para cada chart.

6. Solução de Problemas

6.1. Pods não inicializam

Verifique os logs do Pod com problemas:

Terminal input

```
kubectl logs <pod-name> -n <NAMESPACE>
```

Verifique eventos do namespace:

Terminal input

```
kubectl get events -n <NAMESPACE> --sort-by='.lastTimestamp'
```

6.2. Problemas de autenticação no registry

Recrie o secret de autenticação:

Terminal input

```
kubectl delete secret <docker-registry-secret> -n <NAMESPACE>
kubectl create secret docker-registry <docker-registry-secret> \
  --docker-server=registry.tecnisys.com.br \
  --docker-username=<username> \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  --docker-password=<password> \
  -n <NAMESPACE>
```

EXEMPLO

Exemplo

```
kubectl delete secret regcred -n <NAMESPACE>
kubectl create secret docker-registry regcred \
  --docker-server=registry.tecnisys.com.br \
  --docker-username=<username> \
  --docker-password=<password> \
  -n <NAMESPACE>
```

6.3. Recursos insuficientes

Verifique o consumo de recursos no cluster:

Terminal input

```
kubectl top nodes
kubectl top pods -n <NAMESPACE>
```

Ajuste os limites de recursos no arquivo `values.yaml` e execute um upgrade:

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka
\
  --namespace <NAMESPACE> \
  --values values.yaml
```

EXEMPLO

Exemplo

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade tdp-kafka oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka \
  --namespace tdp-project \
  --values values.yaml
```

Próximos Passos

Após concluir a instalação do TDP Kubernetes, consulte a documentação completa para:

- **Configuração avançada** de componentes
- **Integração** com ferramentas de monitoramento (Prometheus, Grafana)
- **Backup e recuperação** de dados
- **Atualizações** de versões dos componentes
- **Segurança** e controle de acesso (RBAC, Network Policies)

Para mais informações sobre os conceitos e arquitetura do TDP Kubernetes, consulte a seção de Conceitos na documentação.

© Copyright Tecnisys 2026. All rights reserved.

CONCEITOS

© Copyright Tecnisys 2026. All rights reserved.

Kubernetes

Orquestração de Contêineres



> **This component is available only for TDP Kubernetes from v3.0.**

Kubernetes (frequentemente abreviado como **K8s**) é uma plataforma open source de orquestração de contêineres, projetada para automatizar a implantação, o escalonamento e o gerenciamento de aplicações em contêineres.

Originalmente desenvolvido pelo Google e agora mantido pela Cloud Native Computing Foundation (CNCF), o Kubernetes tornou-se o padrão de fato para orquestração de contêineres em ambientes de produção.

No contexto do **TDP Kubernetes**, a plataforma Kubernetes é utilizada como base para a implantação de todos os componentes da edição cloud-native do TDP, substituindo a abordagem tradicional de bare-metal e máquinas virtuais utilizada na edição TDP Datacenter.

Por que Kubernetes?

A adoção do Kubernetes como plataforma de implantação do TDP traz diversos benefícios:

- **Escalabilidade automática:** os componentes podem escalar horizontalmente de acordo com a demanda, sem intervenção manual
- **Alta disponibilidade:** o Kubernetes garante que os componentes sejam redistribuídos automaticamente em caso de falha de um nó
- **Implantação declarativa:** toda a infraestrutura é descrita como código (YAML), permitindo versionamento e reprodutibilidade

© Copyright Tecnisys 2026. All rights reserved.



- **Isolamento de recursos:** cada componente executa em seu próprio contêiner, com limites definidos de CPU e memória
- **Atualizações sem downtime:** rolling updates e rollbacks são nativos da plataforma
- **Ecossistema rico:** integração com ferramentas como Helm, ArgoCD, Prometheus e Grafana

Conceitos Fundamentais

Pods

O **Pod** é a menor unidade de implantação no Kubernetes.

Um Pod encapsula um ou mais contêineres que compartilham o mesmo namespace de rede e armazenamento.

No TDP Kubernetes, cada componente (como Kafka, Trino ou Airflow) é executado em um ou mais Pods.

Elemento	Descrição
Contêineres	Um ou mais contêineres no mesmo Pod (recurso compartilhado)
Rede	Um único IP por Pod; contêineres comunicam via `localhost`
Armazenamento	Volumes compartilhados entre os contêineres do Pod

Deployments e StatefulSets

O Kubernetes utiliza controladores para manter Pods em execução, recriá-los em caso de falha e aplicar atualizações de forma controlada.

- **Deployment:** usado para cargas em que os Pods são intercambiáveis. Um Pod pode ser substituído por outro sem necessidade de preservar o mesmo nome, IP ou volume exclusivo. É comum em componentes como Trino, Superset, CloudBeaver e partes do Airflow, conforme a arquitetura do chart.

© Copyright Tecnisys 2026. All rights reserved.

- **StatefulSet:** usado para cargas que precisam de identidade estável. Cada Pod mantém um nome previsível, uma identidade de rede própria e, quando configurado, um volume persistente associado. É comum em componentes como Kafka, PostgreSQL, ClickHouse e NiFi.

Aspecto	Deployment	StatefulSet
Uso principal	Pods intercambiáveis	Pods com identidade estável
Identidade dos Pods	Não é preservada entre recriações	Preservada entre recriações
Nome dos Pods	Gerado automaticamente	Ordenado e previsível, como `nome-0`, `nome-1`
Armazenamento	Pode usar volumes, mas não há vínculo fixo por Pod	Pode manter volume persistente associado a cada Pod
Escalabilidade	Réplicas equivalentes entre si	Réplicas com identidade própria e ordem controlada
Exemplos no TDP	Trino, Superset, CloudBeaver, partes do Airflow	Kafka, PostgreSQL, ClickHouse, NiFi

Services

Os **Services** expõem os Pods para comunicação interna (ClusterIP) ou externa (NodePort, LoadBalancer, Ingress). No TDP Kubernetes, cada componente possui seu próprio Service para descoberta e comunicação entre componentes.

Tipo	Uso
ClusterIP	Acesso apenas dentro do cluster (padrão); usado para comunicação entre

© Copyright Tecnisys 2026. All rights reserved.

Tipo	Uso
	componentes (ex.: Trino → Hive Metastore).
NodePort	Expõe a aplicação em uma porta em cada nó; útil para testes ou clusters sem LoadBalancer.
LoadBalancer	Expõe via balanceador de carga do provedor de nuvem (ou MetalLB on-premise).
Ingress / Gateway API	Roteamento HTTP/HTTPS por host e caminho; usado para UIs (Airflow, Superset, ArgoCD, etc.).

Ingress vs Gateway API

O TDP Kubernetes pode ser exposto por HTTP/HTTPS usando duas abordagens: Ingress ou Gateway API.

A API Ingress continua disponível e suportada no Kubernetes, mas está congelada: não há previsão de evolução funcional para essa API. Novas funcionalidades de roteamento HTTP/HTTPS estão sendo direcionadas para a Gateway API.

É importante diferenciar a API Ingress dos seus controladores. A aposentadoria anunciada para março de 2026 se refere ao controlador comunitário ingress-nginx, não à remoção da API Ingress do Kubernetes.

Por esse motivo, o TDP Kubernetes documenta as duas abordagens. Ambientes que já utilizam Ingress podem continuar usando essa opção, desde que adotem um controlador compatível e mantido. Para novas implantações ou evoluções de arquitetura, a Gateway API deve ser considerada como a abordagem preferencial.

Namespaces

Os **Namespaces** fornecem isolamento lógico dentro do cluster. O TDP Kubernetes utiliza um namespace único para agrupar os componentes; na documentação esse

© Copyright Tecnisys 2026. All rights reserved.

namespace é indicado como `<NAMESPACE>`.

> **NOTE**
>
> O `<NAMESPACE>` é definido pelo usuário. Pode ser, por exemplo, `tdp-ingestao`, `tdp-financeiro`, `departamento-pessoal` ou qualquer outro nome. O sufixo `-project` em exemplos como `tdp-project` é apenas ilustrativo nesta documentação; o usuário escolhe o nome que desejar.

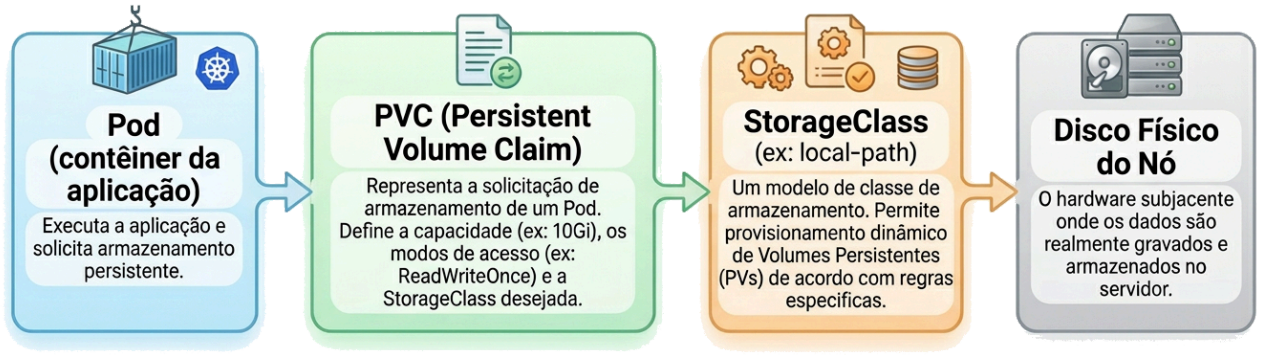
Persistent Volumes

Os **Persistent Volumes (PV)** e **Persistent Volume Claims (PVC)** fornecem armazenamento persistente para os componentes que necessitam manter dados entre reinicializações, como bancos de dados (PostgreSQL, ClickHouse) e sistemas de mensageria (Kafka).

O **PVC** é a solicitação de armazenamento feita pelo Pod (por exemplo, "preciso de 10 Gi em ReadWriteOnce"); o cluster atende a essa solicitação vinculando o PVC a um **PV** existente ou provisionando um novo volume dinamicamente por meio de uma **StorageClass** (por exemplo, `local-path` ou provisionadores de nuvem).

A figura abaixo ilustra esse fluxo: o Pod referencia um PVC; o PVC é atendido pela StorageClass, que provisiona o volume no disco físico do nó (ou em storage externo, conforme a StorageClass configurada).

© Copyright Tecnisys 2026. All rights reserved.



Fluxo de Armazenamento Persistente no Kubernetes.

ConfigMaps e Secrets

- **ConfigMaps:** armazenam configurações não sensíveis em formato chave-valor, utilizadas para parametrizar os componentes
- **Secrets:** armazenam dados sensíveis como senhas, tokens e certificados de forma criptografada

Recurso	Conteúdo típico	Uso no TDP
ConfigMap	URLs, endpoints, flags, arquivos de configuração	Parâmetros de conexão (ex.: catálogos Trino), configs não sensíveis
Secret	Senhas, chaves de API, certificados TLS	Credenciais de banco, registry OCI, LDAP, tokens

Modelo de Deploy do TDP Kubernetes

Helm Charts

© Copyright Tecnisys 2026. All rights reserved.



O **Helm** é o gerenciador de pacotes do Kubernetes, e o TDP Kubernetes utiliza **Helm Charts** para empacotar, configurar e instalar cada componente da plataforma.

Cada chart contém:

- Templates de recursos Kubernetes (Deployments, Services, ConfigMaps, etc.)
- Valores padrão configuráveis (`values.yaml`)
- Declaração de dependências entre charts
- Metadados e documentação

No TDP Kubernetes, o Helm é geralmente adotado em componentes cujo ciclo de vida pode ser tratado adequadamente pelos recursos nativos do Kubernetes, com menor necessidade de automação operacional especializada.

Os charts do TDP Kubernetes estão disponíveis no registry OCI da Tecnisys (`registry.tecnisys.com.br`) e seguem a convenção de nomenclatura `tdp-<componente>` (por exemplo, `tdp-kafka`, `tdp-trino`, `tdp-airflow`).

Registry OCI

No TDP Kubernetes, artefatos como **Helm Charts** e imagens de contêiner podem ser distribuídos por meio de um **registry compatível com OCI**. Esse modelo padroniza o armazenamento, o versionamento e a distribuição dos pacotes utilizados pela plataforma.

O uso de um registry centralizado facilita:

- publicação e distribuição controlada de artefatos;
- versionamento consistente de charts e imagens;
- integração com fluxos de instalação e automação;
- suporte a abordagens baseadas em CLI e GitOps.

Os detalhes de autenticação, acesso ao registry e comandos operacionais devem ser consultados nas seções de instalação.

Operators

Operators estendem o Kubernetes com lógica operacional específica para determinadas aplicações. Eles observam o estado real dos recursos no cluster, comparam esse estado com a configuração desejada e executam ações de reconciliação quando necessário.

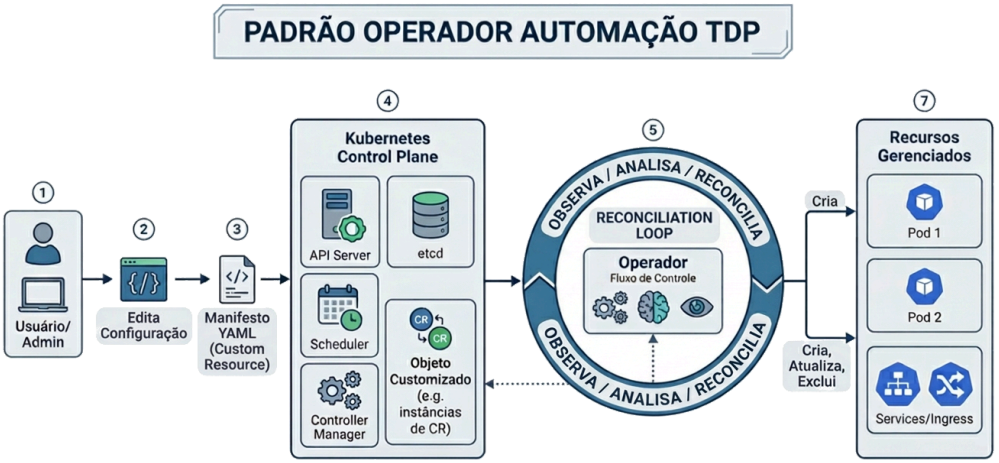
© Copyright Tecnisys 2026. All rights reserved.

Na prática, um Operator automatiza tarefas como criação de recursos derivados, atualização, recuperação de falhas, ajuste de configuração, gerenciamento de réplicas, tratamento de recursos persistentes e manutenção do estado da aplicação.

No TDP Kubernetes, Operators são utilizados nos componentes cuja arquitetura requer reconciliação especializada por meio de Custom Resources. Nesses casos, o Helm permanece como mecanismo de instalação e parametrização do chart, enquanto o Operator é responsável por manter os recursos da aplicação reconciliados no cluster.

Essa abordagem é adotada, por exemplo, em componentes como Kafka, NiFi e ClickHouse, nos quais a operação da aplicação é conduzida por recursos customizados e controladores especializados.

Os CRDs necessários para componentes baseados em Custom Resources são concentrados no chart `tdp-crds`, que deve ser instalado antes dos charts que criam esses recursos.



Padrão Operador Automação TDP.

Estratégia de Deploy: Helm e Operators

A implantação do TDP Kubernetes utiliza o Helm como mecanismo principal de empacotamento, configuração e instalação dos componentes.

Os charts Helm definem os recursos Kubernetes necessários para cada componente, como Deployments, StatefulSets, Services, Secrets, ConfigMaps, Jobs, Ingresses, HTTPRoutes, RBAC e, em componentes baseados em Operators, Custom Resources.

© Copyright Tecnisys 2026. All rights reserved.



Nos componentes em que o TDP Kubernetes adota Operators, o chart Helm entrega os recursos necessários e parametriza a instalação, enquanto o Operator mantém o estado da aplicação reconciliado no cluster.

De forma geral:

- o Helm é o mecanismo de instalação e configuração dos charts do TDP Kubernetes;
- determinados componentes utilizam Operators e Custom Resources por definição da sua arquitetura;
- os CRDs utilizados por esses componentes são instalados previamente pelo chart `tdp-crds`;
- em OpenShift, a implantação segue o modelo Kubernetes, com adaptações de segurança e execução tratadas pelos charts dos componentes quando necessário.

Assim, Helm e Operators atuam de forma complementar no TDP Kubernetes: o Helm entrega a instalação parametrizada, e o Operator, nos componentes que o utilizam, executa a reconciliação operacional da aplicação.

ArgoCD e GitOps

GitOps

GitOps é um modelo operacional no qual o estado desejado do ambiente é definido de forma declarativa em repositórios Git. As mudanças passam a ser versionadas, auditáveis e reconciliadas continuamente no cluster.

Essa abordagem favorece maior previsibilidade operacional, rastreabilidade de mudanças e redução de alterações manuais diretas no ambiente.

Princípios do GitOps

Os princípios centrais do GitOps incluem:

- **Fonte única de verdade**: o estado desejado da plataforma é mantido em repositórios Git;
- **Configuração declarativa**: os recursos são definidos por manifestos e arquivos versionáveis;
- **Versionamento e rastreabilidade**: toda mudança fica registrada e auditável;
- **Reconciliação automática**: o ambiente é continuamente comparado com o estado desejado e ajustado quando necessário;

© Copyright Tecnisys 2026. All rights reserved.

- **Operação previsível**: implantações, mudanças e reversões seguem um fluxo controlado e reproduzível.

ArgoCD

O **ArgoCD** é uma ferramenta de entrega declarativa baseada em princípios de GitOps. Nesse modelo, o estado desejado da aplicação é descrito em repositórios Git, e o ambiente é reconciliado continuamente com esse estado.

No TDP Kubernetes, essa abordagem permite:

- rastreabilidade de mudanças;
- maior previsibilidade de deploy;
- controle declarativo de ambientes;
- redução de alterações manuais diretas no cluster.

Arquitetura do TDP Kubernetes

O TDP Kubernetes pode ser entendido a partir de duas visões complementares.

A primeira é uma **visão lógica**, que mostra como os principais componentes da plataforma se organizam por responsabilidade. Nessa perspectiva, a arquitetura é apresentada em camadas, facilitando a compreensão do papel de cada serviço dentro do ecossistema: interfaces de acesso, motores de processamento, componentes de orquestração, serviços de metadados e segurança, e a base de armazenamento sobre a qual os dados são mantidos.

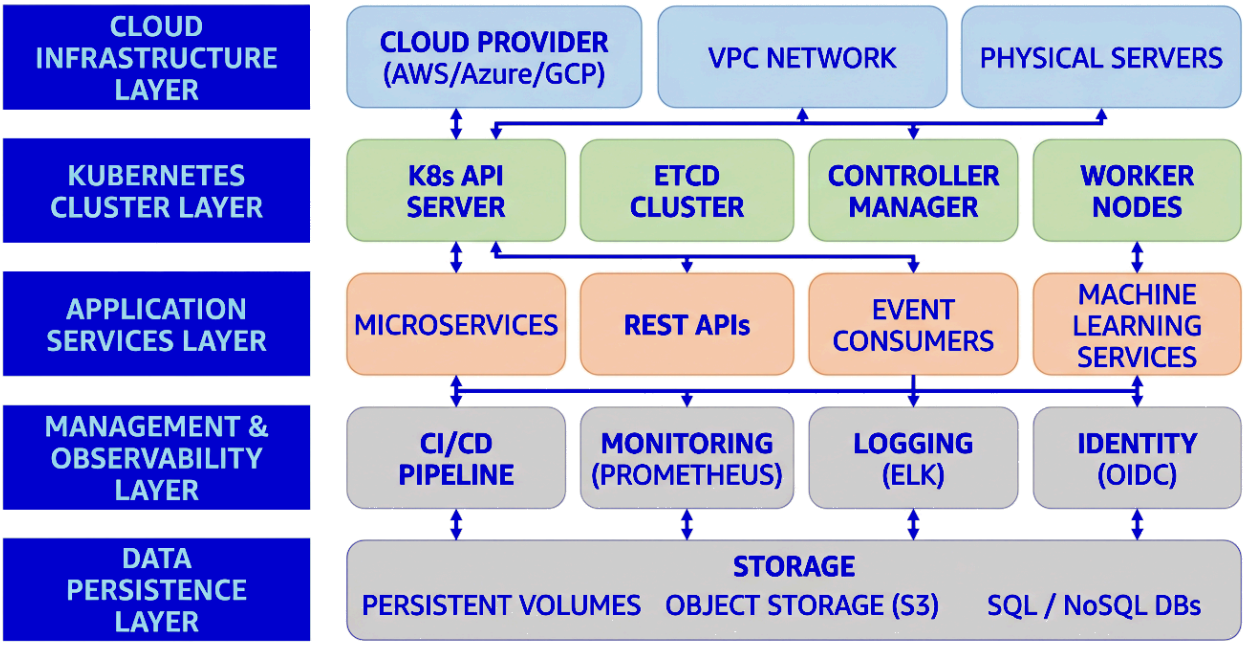
A segunda é uma **visão de implantação**, que mostra como essa arquitetura é materializada em um ambiente Kubernetes. Nessa leitura, o foco deixa de ser apenas a função de cada componente e passa a ser também a forma como artefatos, cluster, persistência, exposição de serviços e operação da plataforma se conectam na prática.

O TDP Kubernetes adota uma arquitetura de microsserviços, em que os componentes do ecossistema Big Data são executados em contêineres gerenciados pelo Kubernetes. Nesse modelo, os Helm Charts encapsulam a complexidade de instalação e configuração dos serviços, enquanto o ArgoCD garante a sincronização contínua entre o estado desejado, definido em Git, e o estado efetivamente aplicado no cluster.

A figura abaixo apresenta a **visão lógica** da plataforma. Ela ajuda a compreender como os serviços se distribuem em camadas funcionais e como cada grupo contribui para o

© Copyright Tecnisys 2026. All rights reserved.

funcionamento do ambiente como um todo.

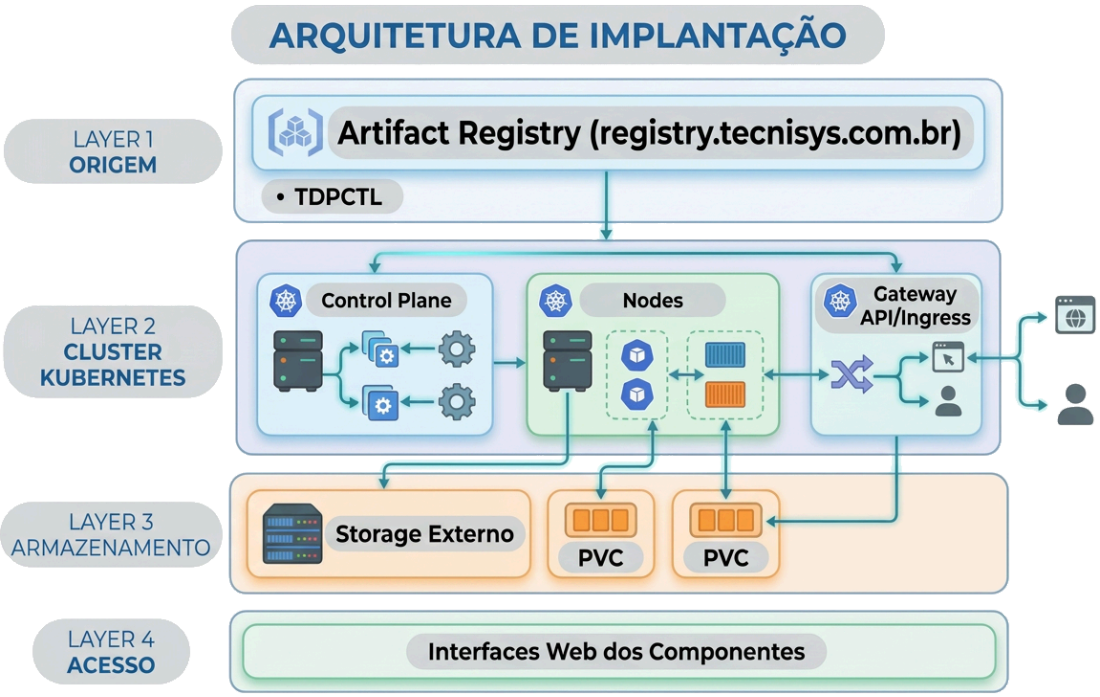


Visão lógica da arquitetura do TDP Kubernetes.

Nessa organização, a camada superior concentra os serviços de acesso e interação com o usuário, como visualização, notebooks, administração e interfaces especializadas. Em seguida, aparecem os motores responsáveis por consultas, processamento analítico, ETL e streaming. A orquestração coordena fluxos e execuções, enquanto a camada de metadados e segurança sustenta catálogo, governança, controle de acesso e persistência auxiliar. Na base, o armazenamento orienta a persistência dos dados e dos formatos analíticos utilizados pela plataforma.

A figura a seguir apresenta a **visão de implantação** do TDP Kubernetes. Ela evidencia como os artefatos são disponibilizados, como os componentes são implantados no cluster e como a plataforma se integra com os mecanismos de persistência e acesso.

© Copyright Tecnisys 2026. All rights reserved.



Visão de implantação do TDP Kubernetes no cluster.

Sob essa perspectiva, o fluxo se inicia na origem dos artefatos, com o registry e as ferramentas de instalação e gestão. Em seguida, a implantação ocorre no cluster Kubernetes, que provê os elementos de controle, execução e exposição dos serviços. O armazenamento persistente dá suporte aos dados e volumes necessários à operação dos componentes, enquanto as interfaces web tornam os serviços acessíveis aos usuários e administradores da plataforma. Assim, a visão de implantação complementa a visão lógica anterior ao mostrar não apenas o papel dos componentes, mas também a forma como eles são distribuídos e operados no ambiente.

Camada	Componentes	Função
Infraestrutura de Plataforma	ArgoCD, PostgreSQL	Suporte à operação da plataforma, gerencia-mento e serviços compartilhados
Mensageria	Apache Kafka	Streaming de eventos e integração de dados

© Copyright Tecnisys 2026. All rights reserved.

Camada	Componentes	Função
Armazenamento de Metadados e Tabelas	Hive Metastore, Delta Lake, Apache Iceberg	Metadados e formatos de tabela abertos
Armazenamento de Objetos	Apache Ozone	Armazenamento distribuído de objetos, alternativa S3-compatível para ambientes on-premise
Segurança	Apache Ranger	Controle de acesso e políticas de segurança
Ingestão	Apache NiFi	Ingestão e transformação de dados
Processamento	Apache Spark, Trino	Processamento batch, streaming e consultas SQL
Orquestração e Desenvolvimento	Apache Airflow, JupyterHub	Orquestração de pipelines e notebooks
Análise e Acesso	ClickHouse, Apache Superset, CloudBeaver	OLAP, visualização e administração de dados
Governança	OpenMetadata	Catálogo de dados e linhagem

Funcionamento da Plataforma

A plataforma TDP Kubernetes pode ser compreendida em um fluxo que vai da definição do estado desejado até a execução dos componentes e o consumo final dos serviços implantados.

1. Controle e Implantação

© Copyright Tecnisys 2026. All rights reserved.

A implantação e a gestão do ambiente podem ocorrer por diferentes abordagens, conforme o modelo operacional adotado:

- **TDPCTL (recomendado):** ferramenta de automação utilizada para reduzir etapas manuais de configuração e implantação.
- **Argo CD com Git:** fluxo GitOps no qual o estado desejado do ambiente é mantido em repositório Git e reconciliado continuamente no cluster.
- **Deploy a partir de artefatos versionados:** os componentes também podem ser implantados a partir de charts e imagens disponibilizados em registry compatível com OCI.

> **NOTE**
>
> Os detalhes operacionais dessas abordagens estão descritos nas seções de instalação e configuração.

2. Distribuição de Artefatos

As imagens de contêiner e os Helm Charts da plataforma são distribuídos por meio de um **registry compatível com OCI**.

Esse modelo permite:

- versionamento centralizado dos artefatos;
- distribuição padronizada de imagens e charts;
- integração com fluxos automatizados de instalação e GitOps;
- controle de acesso conforme o ambiente.

> **NOTE**
>
> Endereço do registry, autenticação, SSO e procedimentos de acesso devem ser tratados nas páginas de pré-requisitos e instalação.

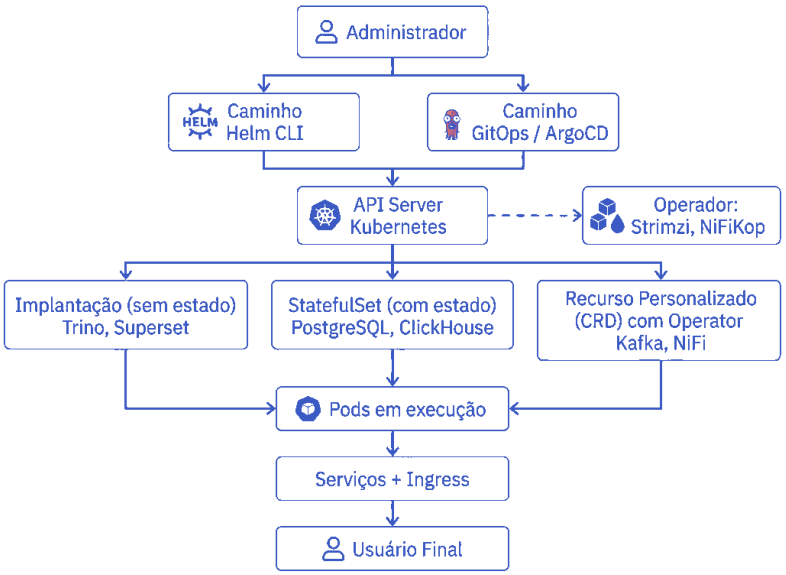
3. Execução dos Componentes

Após a implantação, o cluster Kubernetes executa os componentes da plataforma como workloads conteinerizados, utilizando os mecanismos nativos de agendamento, rede, escalabilidade e persistência.

Nesse modelo:

© Copyright Tecnisys 2026. All rights reserved.

- componentes *stateless* tendem a ser gerenciados por recursos como **Deployments**;
- componentes *stateful* tendem a utilizar **StatefulSets** e volumes persistentes;
- a comunicação entre componentes ocorre por meio de **Services**;
- a exposição de interfaces web pode ser realizada por mecanismos como **Gateway API** ou outras implementações suportadas pelo ambiente.



Modelo de implantação no Kubernetes.

4. Persistência e Acesso aos Dados

A camada de dados da plataforma pode utilizar serviços externos de armazenamento ou recursos providos pela própria solução, conforme a arquitetura do ambiente.

De forma geral:

- serviços compatíveis com **S3** podem ser utilizados quando já existem na infraestrutura;
- em cenários on-premise ou sem storage externo compatível, o **Apache Ozone** pode atuar como camada de armazenamento de objetos;
- componentes que exigem persistência utilizam **Persistent Volumes** e **Persistent Volume Claims**, de acordo com a StorageClass disponível no cluster.

Esse modelo preserva flexibilidade de implantação e compatibilidade entre os componentes da stack.

Próximos Passos

Para seguir com a adoção ou operação do TDP Kubernetes, consulte também:

© Copyright Tecnisys 2026. All rights reserved.

- Pré-Requisitos
- Instalação via Helm
- Instalação via ArgoCD

© Copyright Tecnisys 2026. All rights reserved.

Apache Airflow

Orquestração de Pipelines



Pipeline de dados consiste em um conjunto de tarefas ou ações que precisam ser executadas em determinada ordem ou sequência lógica (*Workflow*) para alcançar um resultado desejado.

Os *pipelines* de dados podem ser representados como DAGs - _Directed Acyclic Graph - que consistem em blocos que seguem uma sequência, à medida em que o anterior é executado, permitindo bifurcações em diversos pontos, não sendo possível retornar ao ponto inicial. Nos DAGs é possível definir operadores (que se transformam em tarefas) e relacionamentos entre operadores de forma programática.

Embora muitos Orquestradores de Pipeline tenham sido desenvolvidos ao longo dos anos para executar tarefas (DAGs), o Airflow possui vários recursos importantes que o tornam especialmente adequado para implementação de *pipelines* de dados eficientes e orientados a *lotes*.

© Copyright Tecnisys 2026. All rights reserved.

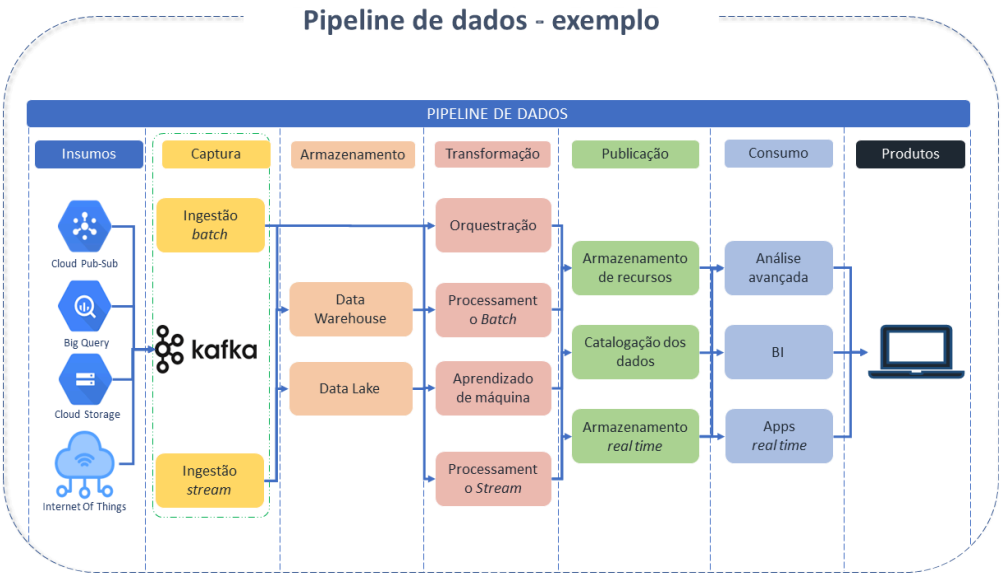


Figura 1 - Pipeline de dados

Características do Apache Airflow:

O Apache Airflow foi criado no Airbnb em 2014 como uma solução para gerenciar seus complexos *Workflows*. Desde o início, o projeto é de *código aberto*, tendo se tornado, em 2017, um projeto *Apache incubator* e, em 2019, um Projeto de nível superior da Apache.

É uma ferramenta rica em recursos e funcionalidades e possui um conjunto de características fundamentais para uma solução de Big Data:

- **Versatilidade**: A capacidade de implementar *pipelines* usando código Python permite criar *pipelines* complexos com qualquer coisa compatível com Python.
- **Fácil Integração**: A base Python do Airflow facilita a extensão e adição de integrações com muitos sistemas diferentes. A comunidade Airflow já desenvolveu uma rica coleção de extensões para diferentes bancos de dados, serviços em nuvens, etc.
- **Rica semântica de programação**: Permite a execução de *pipelines* em intervalos regulares e criação de *pipelines* eficientes usando processamento incremental para evitar recálculos dispendiosos de resultados existentes.
- ***Backfilling***: Permite o reprocessamento de dados históricos de forma fácil, e o recálculo de qualquer conjunto de dados derivados após mudanças no código.
- **Rica Interface Web**: A rica interface web do Airflow fornece visualização fácil para monitoramento de resultados de execuções de *pipeline* bem como a depuração de quaisquer falhas.

© Copyright Tecnisys 2026. All rights reserved.

- **Código aberto**: Garante a criação de trabalhos sem qualquer dependência de fornecedor.

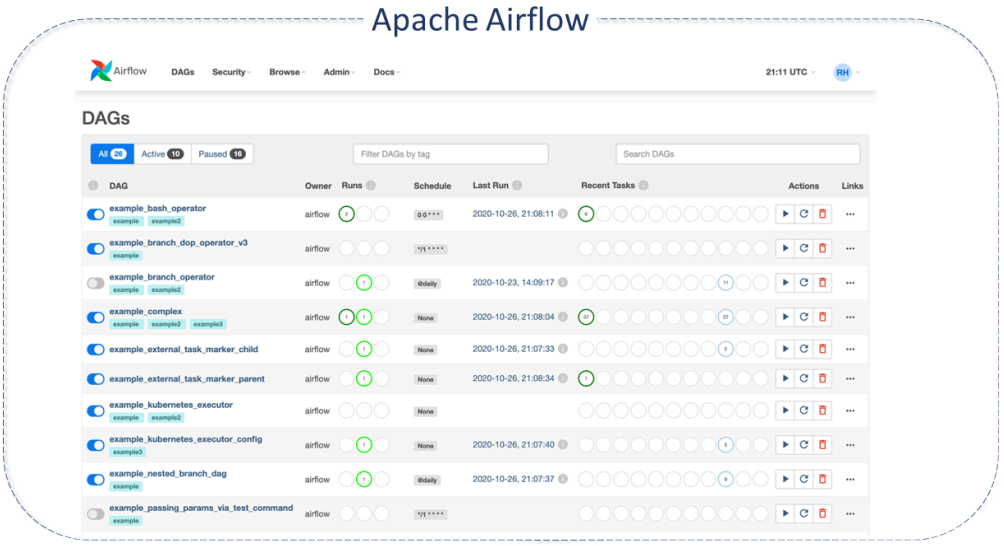
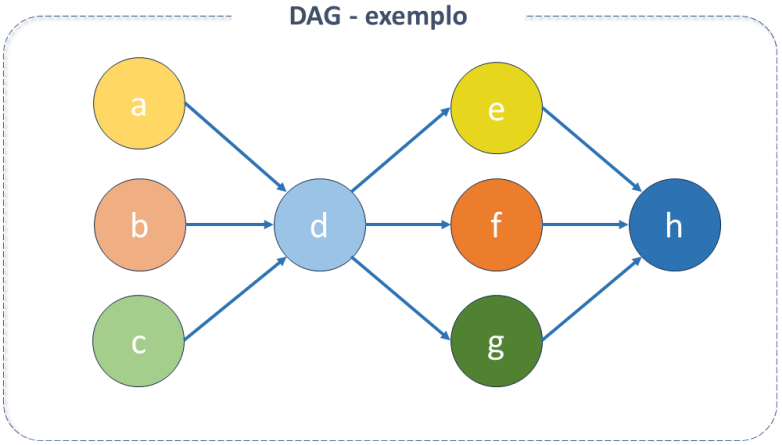


Figura 2 - Interface Apache Airflow

Arquitetura do Apache Airflow

O Airflow é organizado a partir dos seguintes componentes principais:

- **DAGs - _Directed Acyclic Graph: O *DAG* é o conceito principal do Airflow. Representa um *Workflow* (uma coleção de tarefas individuais, organizadas com suas respectivas dependências e fluxos de dados - as próprias tarefas descrevem o que será feito, como busca de dados, análise, acionamento de outro *app*, etc).



© Copyright Tecnisys 2026. All rights reserved.



Figura 3 - Exemplo da estrutura de um DAG

O *DAG* não se preocupa com o que acontece dentro de uma *tarefa*, mas em *como* executá-la (a ordem de execução, quantas vezes repeti-las, se há um tempo limite, etc). A estrutura *DAG* é composta da declaração das dependências entre tarefas. Fora isso, os arquivos *DAG* contêm alguns metadados adicionais sobre o *DAG* informando ao Airflow como e quando deve ser executado. Esta abordagem programática oferece muita flexibilidade para criar *DAGs* e permite personalização na forma de criação dos *pipelines*.
Os *DAGs* não requerem agendamento, mas é muito comum que sejam definidos, o que é feito pelo Scheduler .
Um *DAG* é executado de duas maneiras: acionado manualmente ou por meio de API ou em um cronograma definido, como parte do *DAG*.

Seus parâmetros mais importantes são:

  - **dag_id**: identificador de um *DAG*
  - ***start date***: *Timestamp* a partir do qual o Scheduler escalonará a *DAG*.
  - ***schedule interval***: intervalo de execução da *DAG*.
  - ***default args***: Dicionário com padrões que devem ser passados às tarefas.
- **Scheduler**: O *Scheduler* lida com o acionamento de *workflows* agendados e com o envio de tarefas, para o Executor.

- **Webserver**: Apresenta uma interface de usuário que permite a visualização dos DAGs e tarefas, sua inspeção, acionamento, depuração e análise dos seus resultados.

- **Executor**: Processo que lida com as tarefas em execução. Numa instalação padrão é executado dentro do *Scheduler*. Entretanto, a maioria dos *Executores* adequados para a produção envia a execução da tarefa para os *Workers*.

  A maior parte dos *Executores* irá introduzir outros componentes para que possam "conversar" com seus *Workers* (como a fila de tarefas). Entretanto, podemos entender o *Executor* e seus *Workers* como um componente lógico único lidando com a execução da tarefa.

  O *Executor* é configurável e, dependendo dos requisitos, é possível escolhê-lo dentre algumas opções:

© Copyright Tecnisys 2026. All rights reserved.



- **Locais**:
  - **Sequential Executor**: É o executor *default*. Irá executar uma instância de tarefa por vez.
  - **Local Executor**: Executa tarefas gerando processos de maneira controlada em diferentes modos. Dado que o *BaseExecutor* tem a opção de receber um parâmetro de paralelismo para limitar o número de processos gerados, quando este parâmetro é 0, o número de processos que o *Local Executor* pode gerar fica ilimitado.
- **Remotos**:
  - **Dask Executor**: Permite a execução de tarefas Airflow em um *Cluster Dask Distributed*. *Clusters Dask* podem executar em uma máquina única ou redes remotas.
  - **Celery Executor**: É um dos caminhos para escalar o número de workers.
  - **Kubernetes Executor**: Introduzido no Apache Airflow 1.10.0, permite que o Airflow seja dimensionado com muita facilidade à medida que as tarefas são executadas no Kubernetes.
  - **Celery Kubernetes Executor**: Permite a execução simultânea de *Celery Executor* e *Kubernetes Executor*. Herda a escalabilidade do *Celery Executor* para lidar com altas cargas em horário de pico e isolamento do tempo de execução do *KubernetesExecutor*.
- ***Workers***: São processos separados que executam as tarefas *scheduladas*.
- ***DAG files***: Armazenam os arquivos que são lidos pelo Scheduler e Executor (e pelos *Workers* que o *Executor* tenha).
- **Banco de Dados de metadados**: Banco de dados SQL usado pelo Scheduler, Executor e Webserver para armazenar metadados sobre os *pipelines* de dados que estão sendo executados.

© Copyright Tecnisys 2026. All rights reserved.

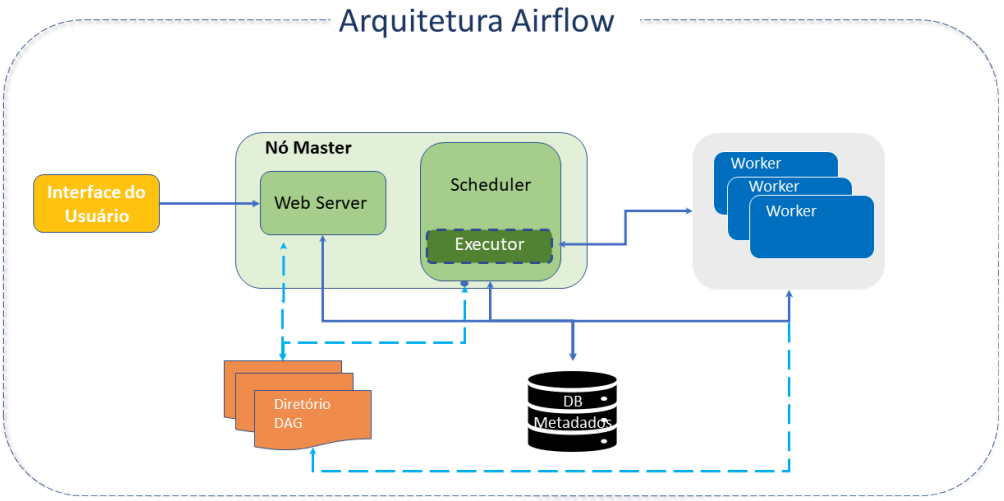


Figura 4 - Arquitetura Airflow

Necessidade de um Diretório Compartilhado para DAGs no Airflow

No ambiente distribuído do Apache Airflow, especialmente quando o executor utilizado é o Celery Executor, Kubernetes Executor ou Dask Executor, é essencial garantir que todos os Workers possuam acesso consistente aos mesmos arquivos de DAG.

As DAGs são os scripts Python que definem os fluxos de tarefas do Airflow.

Quando há múltiplos Workers, cada um executando tarefas em paralelo, todos eles precisam ser capazes de ler o mesmo conjunto de DAGs.

Por padrão, o Scheduler e o Webserver leem os arquivos diretamente do diretório: `/usr/tdp/current/airflow/dags`.

Entretanto, se esse diretório não for compartilhado entre os nós, cada Worker verá apenas os DAGs locais — resultando em falhas na execução ou na ausência dos DAGs na interface Web.

Diretório Compartilhado via NFS

A abordagem mais simples e compatível com o TDP (Tecnisys Data Platform) é montar um NFS (Network File System) exportando o diretório /usr/tdp/current/airflow/dags a partir do nó principal (geralmente onde o Airflow Scheduler está instalado) e montando-o nos demais hosts que executam o Airflow Worker.

© Copyright Tecnisys 2026. All rights reserved.

Dessa forma, todos os componentes do Airflow — Webserver, Scheduler e Workers — acessam exatamente o mesmo repositório de DAGs.

Exemplo de Configuração

1. Instale o servidor NFS e exporte o diretório:

   1.1 Instalar servidor NFS - No servidor NFS (exemplo: tdp-mn01):

   Terminal input

   ```
   sudo yum install -y nfs-utils
   ```

   1.2 Criar diretório de DAGs se não existir:

   Terminal input

   ```
   sudo mkdir -p /usr/tdp/current/airflow/dags
   ```

   1.3 Definir permissões adequadas:

   Terminal input

   ```
   sudo chown -R airflow:airflow /usr/tdp/current/airflow/dags
   sudo chmod 755 /usr/tdp/current/airflow/dags
   ```

2. Edite o arquivo /etc/exports e adicione a linha abaixo (ajuste a rede conforme seu ambiente):

   2.1 Edição do Arquivo:

   Terminal input

   ```
   /usr/tdp/current/airflow/dags  10.0.0.0/24(rw,sync,no_subtree_check)
   ```

   2.2 Exporte e ative o serviço:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
sudo exportfs -arv
sudo systemctl enable nfs-server
sudo systemctl start nfs-server
```

2.3 Verifique se o compartilhamento está ativo:

Terminal input

```
sudo exportfs -v
```

3. Nos nós Airflow Worker (exemplo: tdp-wn01, tdp-wn02):

3.1 Instale o cliente NFS e monte o diretório compartilhado:

Terminal input

```
sudo yum install -y nfs-utils
```

3.2 Crie o diretório de destino (caso não exista):

Terminal input

```
sudo mkdir -p /usr/tdp/current/airflow/dags
```

3.3 Monte o diretório NFS:

Terminal input

```
sudo mount -t nfs tdp-mn01:/usr/tdp/current/airflow/dags
/usr/tdp/current/airflow/dags
```

3.4 Para garantir persistência após reinicialização, adicione a entrada no /etc/fstab:

© Copyright Tecnisys 2026. All rights reserved.



Terminal input

```
    tdp-mn01:/usr/tdp/current/airflow/dags /usr/tdp/current/airflow/dags nfs
defaults 0 0
```

Recursos do Apache Airflow

- **Pooling**: Pools é uma funcionalidade adicional que fornece mecanismos de gerenciamento de recursos. Os pools limitam a execução do paralelismo no recurso quando muitos processos o demandam ao mesmo tempo, evitando sobrecarga.

- **Queuing**: Todas as tarefas vão para a fila padrão. É possível definir filas e workers para consumir tarefas de uma ou mais filas. As filas são especialmente úteis quando algumas tarefas precisam ser executadas em um ambiente ou recurso específico.

- **Plugins**: Airflow tem um gerenciador de *plugins* simples que pode integrar recursos externos ao seu núcleo simplesmente "soltando" arquivos na pasta $Airflow_home/plugins. Os módulos Python na pasta *plugins* são importados e as macros e exibições da web são integradas às principais coleções do Airflow e ficam disponíveis para uso.

Conceitos importantes:

- **Dag Runs**: Toda vez que se executa um DAG, uma nova instância dele é criada, chamada pelo Airflow de *Dag-Run*. As DAG-runs podem ser executadas em paralelo para o mesmo DAG e cada uma tem um intervalo de dados definido que identifica o período de dados e quais tarefas deve operar.

- **Tasks**: Embora do ponto de vista do usuário tarefas e operadores sejam equivalentes, no Airflow existem componentes **tarefas** que gerenciam o estado de operação dos *operators* (definem uma unidade de trabalho dentro de um DAG por meio dos operadores). São representadas como um nó da DAG.

  Existem três tipos comuns de tarefas:

  - **Operators**: São, conceitualmente, um *template* para tarefas pré-definidas que podem ser declaradas dentro do DAG. São os componentes especializados na execução de uma única e específica tarefa dentro do Workflow. É um passo no

© Copyright Tecnisys 2026. All rights reserved.



Workflow e geralmente (não sempre) é atômico, não carregando informações de operadores anteriores. Desta forma detém autonomia. Como são eles que efetivamente executam as tarefas, muitas vezes ambos os termos são utilizados. O Airflow tem um conjunto extenso de **operadores** disponíveis, alguns integrados ao núcleo ou pré-instalados. Os mais populares são:

  - **PythonOperator** chama uma função Python.
  - **EmailOperator**: manda um email.
  - **BashOperator**: executa um script Bash, comando ou conjunto de comandos.
- **Sensors**: Uma subclasse especial de **Operators** que fica à "espera" de um evento externo.

- **Operators**: Uma função customizada Python empacotada como uma tarefa. Torna a criação de DAGs muito mais fácil para quem usa código Python simples em vez de Operators para escrever DAGs.

> **NOTE**
>
> - Se dois operadores precisarem compartilhar informações, podem ser combinados em um único operador. Se isto não for possível há um recurso de comunicação cruzada chamado *xcom*.
> - Operadores não precisam ser atribuídos às DAGs imediatamente (mas, uma vez atribuído, não pode ser transferido ou não-atribuído).

- **Task instances**: Representam o estado de uma tarefa -> em qual etapa do ciclo de vida a tarefa se encontra. Instancia uma tarefa - que foi designada a um DAG e tem um estado associado a uma execução específica. Os estados possíveis são:

  - **None**: A task não foi adicionada à fila de execução (escalonada) pois suas dependêncais ainda não foram supridas.
  - ***Scheduled***: As dependências foram supridas e a tarefa pode ser executada.
  - ***Queued***: a tarefa foi vinculada a um *worker* pelo *executor* e está aguardando disponibilidade para execução.
  - ***Running***: a tarefa está em execução.
  - ***Success***: a tarefa foi executada sem erros.
  - ***Failed***: a tarefa encontrou erros durante a execução e falhou.
  - ***Skipped***: a tarefa foi *bypassada* devido a algum mecanismo.

© Copyright Tecnisys 2026. All rights reserved.



- ***Upstream failed***: as dependências falharam e, portanto, a tarefa não foi executada.
- ***Up for retry***: a tarefa falhou mas existem mecanismos definidos para novas tentativas e, portanto, pode ser re-escalonada.
- ***Sensing***: A tarefa é um *smart sensor*.
- ***Removed***: a tarefa foi removida da DAG desde sua última execução.

Boas Práticas para o Apache Airflow

A comunidade Apache fornece uma série de **Boas práticas**, dentre as quais resumimos algumas a seguir:

- **Escrever um DAG**: Embora a criação de um novo DAG no Airflow seja uma tarefa bem simples, a comunidade alerta para uma série de cuidados necessários para garantir que sua execução não produza resultados inesperados.
- **Gerar DAG dinâmico**: Algumas vezes escrever um DAG manualmente não é prático. O recurso de geração dinâmica de DAGs pode ser útil nestas ocasiões.
- **Acionamento de DAGs após alterações**: O sistema necessita de tempo suficiente para processar arquivos alterados. É recomendável que se evite acionar DAGs logo após a sua alteração ou a alteração de arquivos que o acompanhem.
- **Reduzir a Complexidade do DAG**: Embora o Airflow seja bom para lidar com grandes volumes de DAGs com muitas tarefas e dependências, DAGs muito complexos e em grande quantidade podem afetar o desempenho do Scheduler. É recomendável simplificá-los e otimizá-los sempre que possível.
- **Testar um DAG**: Os DAGs devem ser tratados como código de nível de produção e possuir inúmeros testes associados, para garantir que produzam os resultados desejados. É possível escrever uma grande variedade de testes para um DAG.
- **Manutenção do DB Metadados**: Com o passar do tempo, o banco de Metadados aumenta sua área de cobertura de armazenamento. O CLI do Airflow permite limpar dados antigos com o command airflow db clean.
- **Manipular dependências conflitantes/complexas do Python**: O Airflow possui muitas dependências do Python que, às vezes entram em conflito com o que o código deseja. A comunidade oferece algumas estratégias que podem ser empregadas para mitigar os riscos.

Detalhes do Projeto Apache Airflow

O Airflow foi escrito em Python. Seus fluxos de trabalho são criados por meio de *scripts* Python. Foi projetado sob o princípio de "configuration as code". Embora existam outras

© Copyright Tecnisys 2026. All rights reserved.

ferramentas que adotem este princípio, o uso do Python é o que permite aos desenvolvedores importação de bibliotecas e classes.

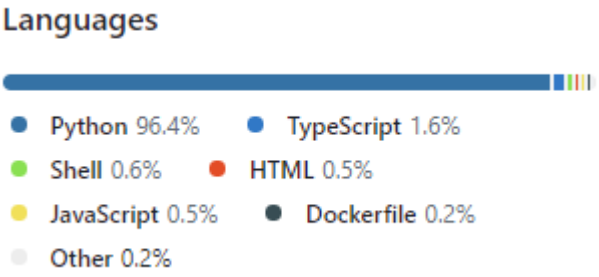


Figura 5 - Linguagens do Airflow

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **3.0.2**, distribuída pelo Helm Chart `tdp-airflow` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fonte(s):

- Airflow.apache.org.
- cwiki.apache.org.
- github.com/apache/airflow
- Airflow-fundamentos (comunidade).

© Copyright Tecnisys 2026. All rights reserved.

Apache Ambari

Gestão do Cluster



Desenvolvido para facilitar a implantação e administração de *Clusters* de *Big Data*, o Apache Ambari possibilita a automatização do *deploy*, gerenciamento de serviços e nós (*hosts*), monitoramento sistêmico do ambiente, versionamento de configurações e muito mais.

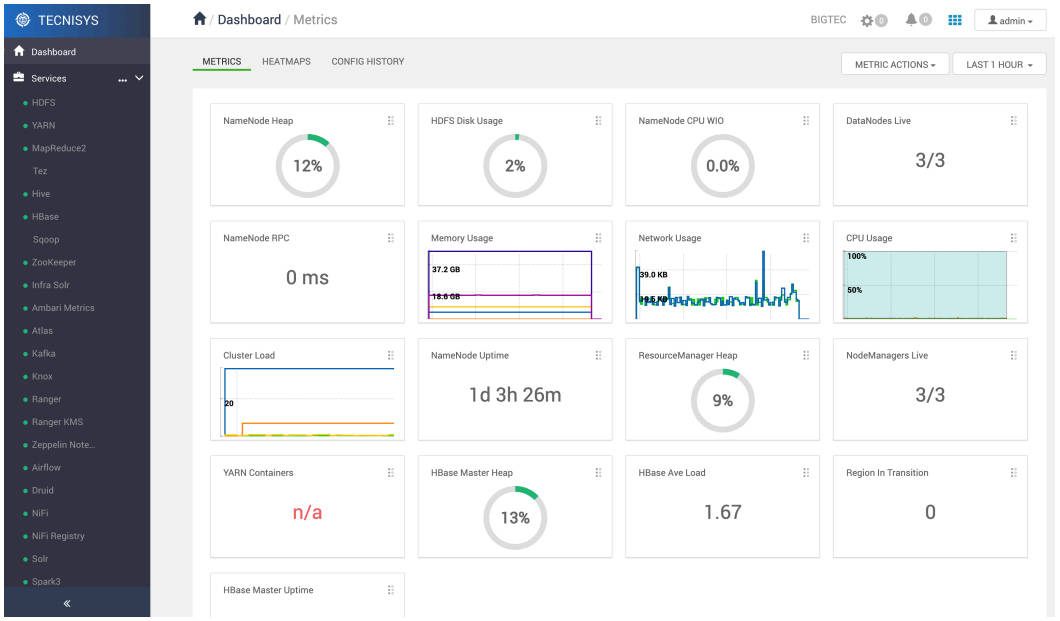


Figura 1 - Exemplo do Ambari

Sua arquitetura consistente, API REST robusta e interface web intuitiva e interativa fornecem todos os recursos necessários para a administração centralizada do *Cluster*.

Atualmente, o projeto Apache Ambari está a todo vapor graças a uma comunidade ativa e renovada. É possível acompanhar essa evolução constante no seu repositório no Github.

© Copyright Tecnisys 2026. All rights reserved.



Objetivos

Dentre os seus principais objetivos, idealizados no início do projeto, destacam-se:

- **Plataforma Independente:** O sistema deve suportar arquiteturalmente qualquer hardware e sistema operacional. Componentes que são inerentemente dependentes de uma plataforma devem ser conectáveis com interfaces bem definidas.
- **Componentes Plugáveis:** A arquitetura não deve assumir ferramentas e tecnologias específicas. Quaisquer ferramentas e tecnologias específicas devem ser encapsuladas por componentes conectáveis. A arquitetura deve ser facilmente extensível. A meta de plugabilidade não abrange a padronização de protocolos entre componentes ou interfaces para operar com implementações de terceiros.
- **Gerenciamento de Versão e Atualização:** Os componentes, em execução em vários nós (*hosts*), devem suportar diferentes versões de protocolos para assim suportarem atualizações independentes. A atualização de qualquer componente do Ambari não deve afetar o estado do *Cluster*.
- **Extensibilidade:** A arquitetura deve suportar a adição de novos serviços, componentes e APIs. A extensibilidade também implica facilidade na modificação de qualquer configuração ou etapas de provisionamento para Plataformas de Serviços. Além disso, a possibilidade de suportar Plataformas de Serviços diferentes precisa ser considerada.
- **Recuperação de Falha:** O sistema deve ser capaz de se recuperar de qualquer falha de componentes para um estado consistente. O sistema deve tentar concluir as operações pendentes após a recuperação. Se certos erros são irrecuperáveis, a falha ainda deve manter o sistema em um estado consistente.
- **Segurança:** A segurança implica 1) autenticação e autorização baseada em perfis de usuários (API e interface web), 2) instalação, gerenciamento e monitoramento da Plataforma de Serviços protegida via Kerberos e 3) autenticação e criptografia da comunicação na rede entre os componentes do Ambari.
- **Rastreamento de Erro:** A arquitetura deve se esforçar para simplificar o processo de rastreamento de falhas. As falhas devem ser propagadas para o usuário com os detalhes e indicadores necessários para a análise.
- **Feedback Intermediário e Operações em Near Real Time:** Para operações que demoram um pouco para serem concluídas, o sistema precisa ser capaz de fornecer feedback ao usuário com progresso intermediário em relação à execução atual de tarefas, percentual de operação concluída e referência ao log de operação, em tempo hábil (quase em tempo real).

© Copyright Tecnisys 2026. All rights reserved.

Terminologia

Seguem os significados dos termos técnicos do projeto Apache Ambari utilizados nesta documentação:

- Serviço (Service): Serviço refere-se aos serviços da Plataforma de Serviços, como HDFS, YARN, Spark, Kafka, entre outros. Um serviço pode ter vários componentes (por exemplo, o HDFS possui NameNode, DataNode e etc.) ou ser apenas uma biblioteca cliente, sem nenhum serviço de daemon em execução contínua no cluster.

- Componente (Component): Um serviço consiste de um ou mais componentes. Por exemplo, o HDFS possui 3 componentes: NameNode, NameNode Secundário e DataNode. Os componentes podem ser opcionais. Um componente pode abranger vários nós (por exemplo, instâncias do componente DataNode em vários nós).

- Nó (Node ou Host): Nó refere-se a uma máquina (física ou virtual) no Cluster. Nó e host são usados de forma intercambiável nesta documentação.

- Operação (Operation): Uma operação refere-se a um conjunto de mudanças ou ações executadas em um cluster para satisfazer uma solicitação do usuário ou para obter uma mudança de estado desejável no Cluster. Por exemplo, iniciar um serviço ou executar um smoke test são operações. Se uma solicitação do usuário para adicionar um novo serviço o Cluster inclui a execução de um smoke test também, todo o conjunto de ações para atender à solicitação do usuário irá compor uma Operação. Uma operação pode consistir de em múltiplas “ações” ordenadas.

- Tarefa (Task): Tarefa é a unidade de trabalho enviada para execução em um nó. Uma tarefa é o trabalho que o nó tem que realizar como parte de uma ação. Por exemplo, uma “ação” pode se composta pela instalação de um DataNode no nó N1 e a instalação de um DataNode e um NameNode Secundário no nó N2. Neste caso, a “tarefa” para N1 será instalar um DataNode e as “tarefas” para N2 serão instalar um DataNode e um Namenode Secundário.

- Estágio (Stage): Um estágio refere-se a um conjunto de tarefas necessárias para concluir uma operação e são independentes entre si. Todas as tarefas no mesmo estágio podem ser executadas em diferentes nós em paralelo.

© Copyright Tecnisys 2026. All rights reserved.

- Ação (Action): Uma "ação" consiste em uma tarefa ou tarefas em uma máquina ou um grupo de máquinas. Cada ação é rastreada por um ID e os nós relatam o status da mesma pelo menos no granularidade da ação. Uma ação pode ser considerada uma etapa em execução. Nesta documentação, um estágio e uma ação têm correspondência de um para um, a menos que especificado de outra forma. Um ID de ação será uma bijeção de request-id para stage-id.

- Plano de Estágio (Stage Plan): Uma operação normalmente consiste em várias tarefas em várias máquinas e elas geralmente possuem dependências que exigem que sejam executadas em uma ordem específica. Algumas tarefas devem ser concluídas antes que outras possam ser agendadas. Portanto, as tarefas necessários para uma operação podem ser divididas em várias etapas, na qual cada etapa deve ser concluída antes do próximo estágio, mas todas as tarefas no mesmo estágio podem ser programadas em paralelo em diferentes nós.

- Manifesto (Manifest): O manifesto refere-se à definição de uma tarefa que é enviada a um nó para execução. O manifesto deve definir completamente a tarefa e deve ser serializável. O manifesto também pode ser persistido no disco para recuperação ou registro.

- Função (Role): Uma função mapeia um componente (por exemplo, NameNode) ou uma ação (por exemplo, rebalanceamento do HDFS, smoke test do HBase e etc.).

Arquitetura

A arquitetura do Ambari envolve:

- **Ambari Web:** Aplicação *client-side* responsável pela *interface web* que exibe informações e executa operações no *Cluster* por meio da API REST fornecida pelo Ambari Server.
- **Ambari Server:** Componente Master responsável pelo controle *Cluster*. Consiste de vários *entry-points* disponíveis para diferentes necessidades, tais como:
  - Gerenciamento do *Daemon*: *entry-point* para *start*, *stop*, *reset* e *restart* do *daemon* do Ambari Server.
  - Atualização de Software: *entry-point* para *upgrade* do Ambari Server após a sua instalação.
  - Configuração de Software: *entry-point* para *setup* preliminar do Ambari.
  - Configuração de Autenticação e Autorização: *entry-point* para *setup* do mecanismo de gerenciamento de identidade, autenticação e autorização.

© Copyright Tecnisys 2026. All rights reserved.

Opções disponíveis: LDAP (Lightweight Directory Access Protocol), PAM (Pluggable Authentication Module) e Kerberos.
    - Backup e Restore: *entry-point* para a realização de *backup* (*snapshot*) e *restore* da instalação atual, com exceção do banco de dados.
- **Ambari Agent:** Componente Slave responsável pela execução de ações e envio de informações e métricas de determinado *host*.
- **Banco de Dados:** Área de persistência do estado, métricas e metadados da infraestrutura do *Cluster* (serviços e *hosts*). O Ambari oferece suporte a vários Sistemas Gerenciadores de Banco de Dados Relacional (SGBDR), tais como PostgreSQL (*default*), MySQL, MariaDB, Oracle e Microsoft SQL Server, cuja a escolha é feita durante o primeiro *setup* do Ambari Server.

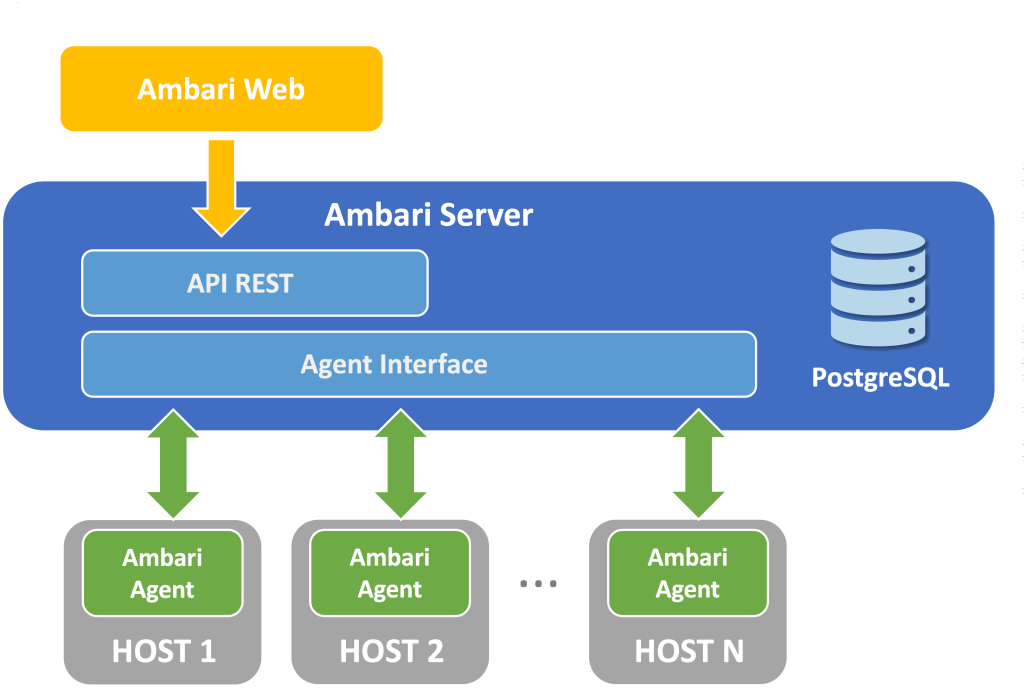


Figura 2 - Arquitetura do Ambari

Detalhes sobre o projeto

O Ambari é desenvolvido, predominantemente, em Java, JavaScript e Python.

© Copyright Tecnisys 2026. All rights reserved.

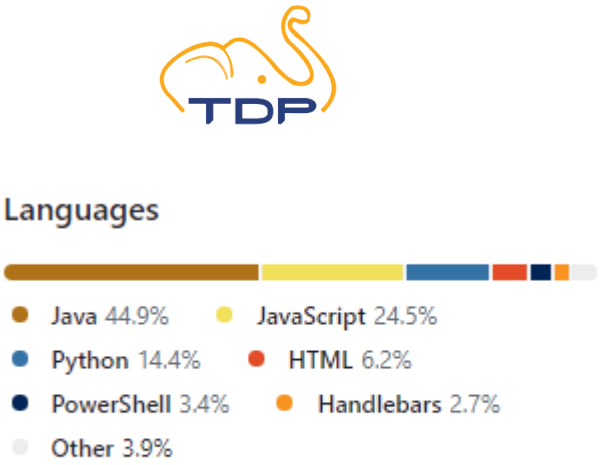


Figura 3 - Linguagens predominantes do projeto Apache Ambari

Fontes:

- https://ambari.apache.org
- https://github.com/apache/ambari

© Copyright Tecnisys 2026. All rights reserved.

Apache Atlas

Governança de Dados



Em resposta à intensa busca por valor e velocidade na tomada de decisão pelas organizações modernas, o mercado oferece, cada vez mais, tecnologias, ambientes e componentes para serem consumidos e explorados com estratégias *realtime*, *near-time*, *streaming* e outras que permitam a transformação dos dados e o acesso às informações no menor tempo possível.

Embora esta evolução analítica agregue alta capacidade às transformações digitais e otimize os parques de tecnologia, demanda das organizações um "alicerce" estruturado e alta maturidade na manutenção de processos e ativos.

É neste momento que entra a Governança de Dados, processo fundamental que garante alinhamento à conformidade das organizações, melhorando a qualidade das percepções, auxiliando na adoção da conformidade regulatóri1, na redução de custos com políticas e sistemas centralizados, no crescimento dos dados controlados e organizados, protegendo-os de qualquer contratempo, dentro de um ambiente cultural favorável à inovação e permeando todo o *pipeline* de tecnologias em que os dados navegarão.

© Copyright Tecnisys 2026. All rights reserved.

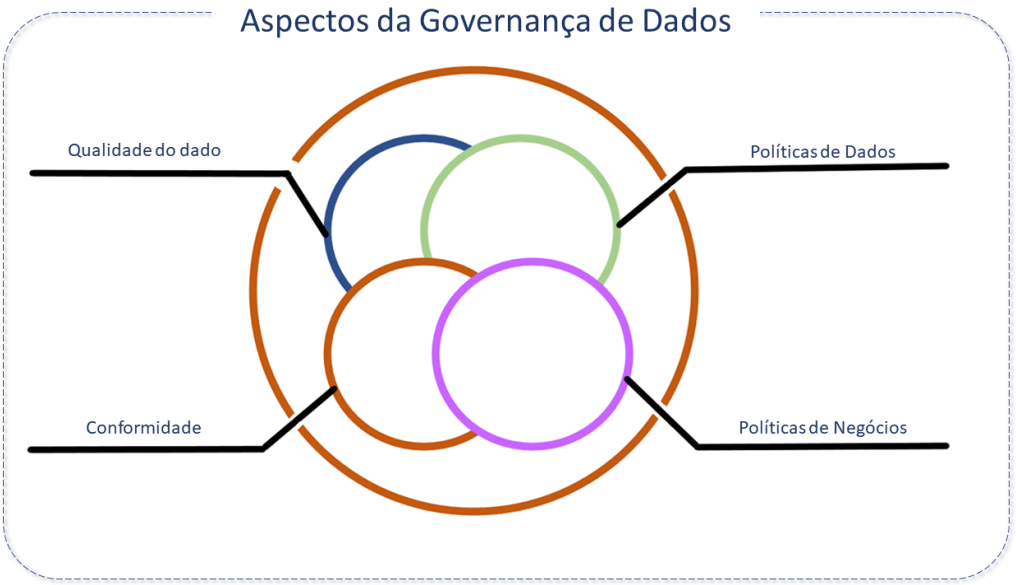


Figura 1 - Aspectos da Governança de dados

Características do Apache Atlas

O Apache Atlas é um conjunto escalável de serviços básicos de governança fornecidos para o Hadoop, com o objetivo de auxiliar as organizações na realização de seus requisitos de conformidade. Para isso, usa modelos prescritivos e *forenses* enriquecidos por metadados *taxonômicos* de negócios.

O Apache Atlas permite que as organizações construam um catálogo de seus bens, classificando-os, administrando-os e provendo competências de colaboração para seu uso por cientistas de dados e times de governança.

A ferramenta foi projetada para trocar metadados com outras ferramentas e processos dentro e fora da pilha Hadoop, permitindo assim controles de governança independentes de plataforma atendendo, com eficácia, aos requisitos de conformidade.

Esses serviços incluem:

- **Pesquisa e linhagem proscritiva:** facilitando a exploração pré-definida e *ad hoc* de dados e metadados e mantendo um histórico de fontes de dados e de como dados específicos foram gerados.
- **Controle de acesso a dados orientado por metadados.**
- **Modelagem flexível de dados comerciais e operacionais.**
- **Classificação de dados:** auxiliando no entendimento da natureza dos dados e sua classificação com base em fontes externas e internas.

© Copyright Tecnisys 2026. All rights reserved.

- **Intercâmbio de metadados com outras ferramentas de metadados.**

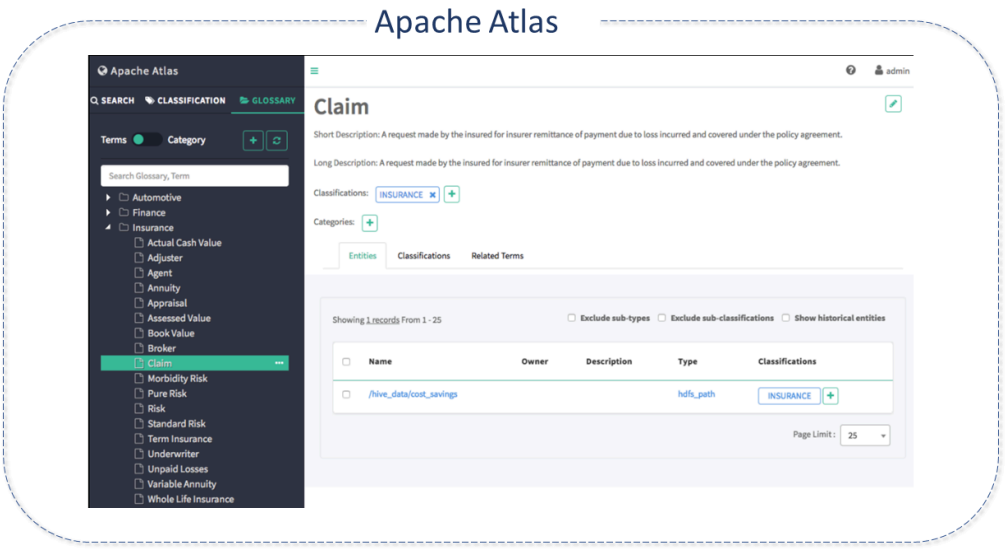


Figura 2 - Interface Apache Atlas

Arquitetura do Apache Atlas

A arquitetura do Apache Atlas envolve os seguintes componentes:

Core

- Type System: Permite a definição de um modelo para objetos de metadados que se deseja gerenciar.
    - O modelo é composto de definições chamadas *types*.
    - As instâncias das *types* são chamadas *entidades* e representam os objetos de metadados que serão gerenciados.
    - Todos os objetos de metadados gerenciados pelo Atlas no modelo *out-of-the-box* (prontos para uso) são modelados usando *types* e representados como entidades.
    - A natureza genérica da modelagem no Atlas permite que integradores e administradores de dados definam metadados técnicos e de negócio e os relacione por meio de recursos do Atlas.
- **Engine Graph:** Internamente, Atlas persiste os objetos de metadados que gerencia usando um modelo *Graph*. Esta abordagem provê grande flexibilidade e habilita a manipulação eficiente dos relacionamentos entre os objetos de metadados.

© Copyright Tecnisys 2026. All rights reserved.



    - A engine *Graph* é responsável pela tradução entre *types* e *entidades* do *Type System* e pelo modelo de persistência do gráfico subjacente.
    - A engine *Graph* também cria índices para objetos metadados de forma que se possa executar buscas de forma eficiente.
    - O Atlas usa o JanusGraph para armazenar objetos metadados.
- **Ingestão/Exportação:** O componente *ingest* habilita a adição de metadados no Atlas e o componente *export* disponibiliza as mudanças detectadas em metadados para serem geradas como eventos e consumidas pelos *Consumers* em resposta às mudanças em metadados em tempo real.

Integração

Existem dois métodos para gerenciar metadados no Atlas:

- **API:** Todas as funcionalidades do Atlas são disponibilizadas ao usuário final via REST API, que habilita a criação, alteração e exclusão de *types* e *entidades*. É este o principal mecanismo para consulta e *discover* dos *types* e *entidades* gerenciados pelo Atlas.

- **Mensagens:** Adicionalmente às APIs, há a possibilidade de integração com o Atlas usando a interface *messaging*.

    - Esta interface é muito útil para a comunicação entre objetos de metadados para o Atlas e para consumir eventos de alteração de metadados do Atlas. É particularmente útil quando se deseja usar uma integração mais flexível para alcançar mais escalabilidade, confiança, etc.
    - O Atlas usa o Kafka como servidor de notificações para comunicação entre consumidores *hooks* e *downstream* de eventos de notificação de metadados. Eventos são escritos pelo *hooks* e Atlas para diferentes tópicos Kafka.
- **Fontes de metadados**

  O Atlas oferece suporte a integração com diversas fontes de metadados, prontas para uso. A integração implica em duas coisas:

    - Existirem modelos de metadados que Atlas define nativamente para representar objetos destes componentes.
    - Existirem componentes fornecidos pelo Atlas para "ingerir" objetos metadados desses componentes (real time ou batch).

© Copyright Tecnisys 2026. All rights reserved.

Atualmente, suporta a ingestão e gerenciamento de metadados das seguintes fontes:

  - HBase
  - Hive
  - Sqoop *(obsoleto — aposentado pela Apache Software Foundation)*
  - Storm
  - Kafka
  - Falcon

Aplicações

Os metadados gerenciados pelo Apache Atlas são consumidos por uma variedade de aplicações.

- **Admin UI:** Este componente, baseado na Web, habilita administradores e cientistas de dados a "descobrir" e "anotar" metadados.
  - É uma interface de pesquisa com linguagem de consulta semelhante a SQL que pode ser usada para consultar os tipos de metadados e objetos gerenciados pelo Atlas.
  - A Interface usa a API REST do Atlas para criar sua funcionalidade.
- **Políticas baseadas em *tags* do Ranger:** O Ranger é uma solução avançada de gerenciamento de segurança para o ecossistema Hadoop com ampla integração com uma variedade de componentes.
  - Ao integrar-se com o Atlas, permite que administradores de segurança definam políticas orientadas por metadados, visando uma governança eficaz.
  - É um *Consumer* de eventos de alteração de metadados notificados pelo Atlas.

© Copyright Tecnisys 2026. All rights reserved.

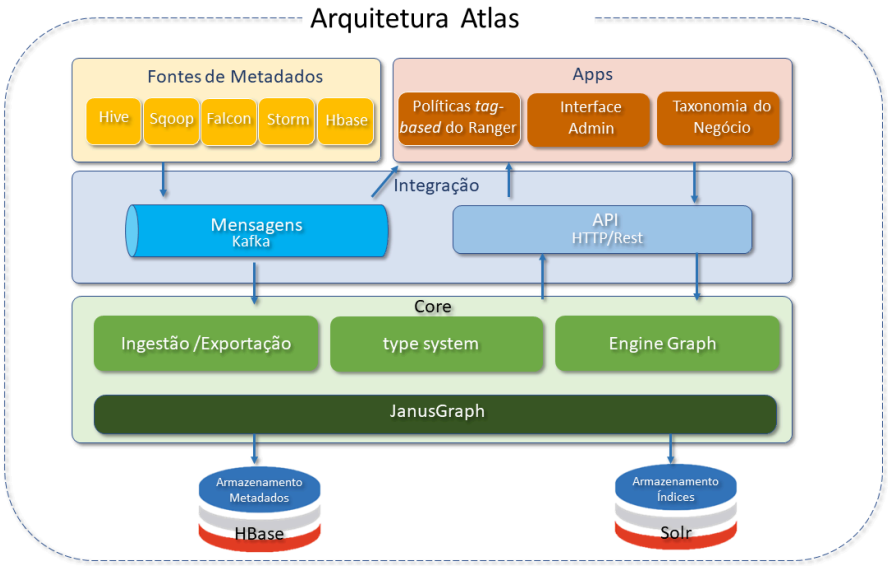


Figura 3 - Arquitetura Apache Atlas

Recursos do Apache Atlas

- **Base de conhecimento:** Aproveita os *metadados* existentes. Suporta a troca de metadados com outros componentes, aplicativos de terceiros ou ferramentas de governança.

- **Auditoria centralizada:** Armazenamento indexado e "pesquisável" a partir de repositório histórico de todos os eventos de governança, incluindo acessos, concessões, negações, eventos operacionais relacionados à proveniência de dados e métricas.

- ***Engine* de política:** Política de conformidade em tempo de execução com base em esquemas de classificação de dados, atributos e funções.

- ***Interface RESTful*** (em conformidade com os critérios REST): Suporte a aplicativos de terceiros por meio de APIs REST.

- **Linhagem**

  - UI intuitiva para visualizar linhagens de dados, à medida em que se movem pelos processos.
  - APIs REST para trabalhar com tipos e instâncias.
- **Integração com Ranger:**: Integração com Apache Ranger habilitando autorização/mascaramento de dados no acesso ao dado, baseado em classificações

© Copyright Tecnisys 2026. All rights reserved.

associadas com entidades. Pode ser usada para implementar políticas de segurança baseadas em classificação dinâmica e em papéis.

- **Alta disponibilidade e Tolerância a falhas:** O Apache Atlas usa e interage com uma variedade de sistemas para prover gerenciamento de metadados e linhagem de dados para administradores de dados. Estas dependências devem ser configuradas adequadamente, para que seja possível alcançar um alto grau de disponibilidade. A comunidade descreve este suporte em detalhes aqui.

- **Propagação da Classificação:**: Apache atlas habilita classificações para entidades serem associadas automaticamente com outras entidades relacionadas. Isto é muito útil quando lidamos com cenários onde um *dataset* deriva seu dado a partir de outros *datasets*.

- **Metadados do Negócio**: O *typesystem* do Atlas permite a definição de um modelo e a criação de entidades para objetos de metadados que se deseja gerenciar. O modelo captura atributos técnicos como nome, descrição, data de criação, número de réplicas, etc. Isto é muito útil para expandir atributos técnicos com atributos adicionais da captura de detalhes do negócio para auxiliar na organização, busca, gerenciamento de entidades de metadados.

- **Segurança:**: Os seguintes recursos de segurança estão disponíveis para auxiliar na segurança da Plataforma:

    - Suporte a SSL one-way (autenticação do servidor) e two-way (autenticação do cliente e servidor).
    - Autenticação de serviços: A Plataforma, ao iniciar, é associada a uma identidade autenticada. Por default, em um ambiente não assegurado, esta identidade é a mesma que o usuário autenticado no S.O. para iniciar o servidor. No entanto, em um *Cluster* seguro, que utiliza Kerberos, é uma prática recomendada configurar um *Keytab* e *Principal* para que a Plataforma seja autenticada no *KDC*. Com isso, o serviço interagirá com outros serviços de *Cluster* seguros.
    - Autenticação baseada em SPNEGO: Os acessos HTTP à Plataforma Atlas podem ser protegidos ativando o suporte SPNEGO da plataforma. Existem atualmente dois mecanismos suportados:
        - simples: a autenticação é realizada por meio de um nome de usuário fornecido.

© Copyright Tecnisys 2026. All rights reserved.

    - Kerberos: a identidade autenticada *KDC* do cliente é aproveitada para autenticar no servidor.
- **Índice de Reparo**:: Em casos raros, durante a criação da entidade, os índices correspondentes não são criados no Solr. Como o Atlas depende muito do Solr, isto resultaria na entidade não retornada por uma pesquisa (a pesquisa avançada não é afetada). O Atlas Index Repair Utility para JanusGraph permite a restauração de todos os índices.

- **Tipo de Entidade *Atlas Server***: O tipo de entidade AtlasServer é um tipo especial de entidade:

    - Criada durante operações de *Export* ou *Import*.
    - Possui páginas de propriedades especiais que mostram auditorias detalhadas para operações de importação e exportação.
    - Entidades são vinculadas a ela, usando a nova opção dentro do atributo de entidade *SoftReference*.
- **Atributos replicados**:: Por meio dos atributos do tipo de entidade *Referenciável* *replicatedFrom* e *replicatedTo* é possível saber como entidades chegaram à instância do Apache Atlas, se foram criadas por ingestão de gancho ou importadas de outra instância.

- ***SoftReference***:: A estratégia de persistência do atributo é determinada baseada em seu tipo. A opção do atributo *isSoftReference* definida como *true* faz com que um tipo de atributo não-primitivo receba o tratamento de atributo primitivo.

- **Políticas de acesso Ranger-Atlas**::

    - **Controles de acesso baseados em classificação:** Uma entidade de dados pode ser marcada como metadados relacionada à conformidade ou determinada taxonomia e usada para atribuir permissões a um usuário ou grupo.
    - **Política de acesso baseada em expiração de dados:** Datas de expiração para negar acesso automaticamente aos dados marcados após a data indicada.
    - **Políticas de acesso específicas do local:** Acesso a um usuário enquanto estiver em determinado local, mas não em outro, embora se trate do mesmo usuário.
    - **Proibição de combinações de conjuntos de dados:** Restrição baseado em conjuntos de dados (como por exemplo, para evitar que sejam combinados).

Detalhes do Projeto Apache Atlas

© Copyright Tecnisys 2026. All rights reserved.

Apache Atlas foi desenvolvido predominantemente em Java. Possui componentes de *Dashboard* em JavaScript.

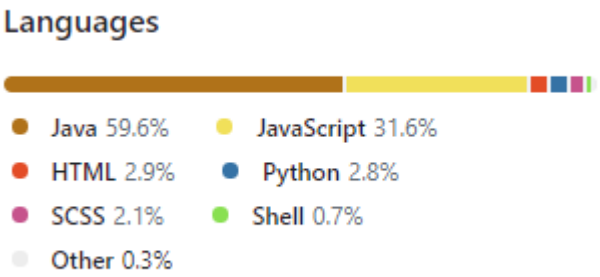


Figura 4 - Linguagens do Atlas

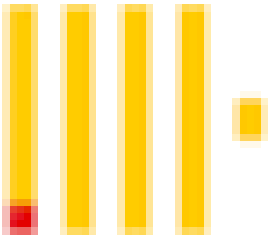
Fontes:

- Atlas.apache.org
- GitHub - Apache Atlas
- Guia do Usuário técnico - Apache Atlas

© Copyright Tecnisys 2026. All rights reserved.

ClickHouse

Banco de Dados OLAP Colunar



> **!** This component is available only for TDP Kubernetes from v3.0.

O **ClickHouse** é um sistema de gerenciamento de banco de dados colunar (DBMS) open source, de alto desempenho, projetado para processamento analítico online (OLAP). Desenvolvido originalmente pela Yandex para o serviço de análise web Yandex.Metrica, o ClickHouse é capaz de processar bilhões de linhas e dezenas de gigabytes de dados por segundo em um único servidor.

No TDP Kubernetes, o ClickHouse é um componente **exclusivo** da edição Kubernetes, oferecendo uma solução de análise de dados em tempo real complementar ao Apache Superset e ao Trino.

Por que ClickHouse?

O ClickHouse se destaca em cenários analíticos de alto volume por diversas razões:

- **Performance extrema:** consultas analíticas em bilhões de registros em milissegundos
- **Armazenamento colunar:** compressão eficiente e leitura otimizada para consultas que acessam poucas colunas
- **Escalabilidade linear:** distribuição de dados e consultas em múltiplos nós do cluster
- **SQL compatível:** suporta uma sintaxe SQL rica, facilitando a adoção por analistas e engenheiros de dados
- **Ingestão em tempo real:** capaz de inserir milhões de registros por segundo
- **Compressão inteligente:** algoritmos de compressão adaptados ao tipo de dado de cada coluna

© Copyright Tecnisys 2026. All rights reserved.

Armazenamento Colunar vs. Armazenamento em Linhas

A principal diferença entre o ClickHouse e bancos de dados tradicionais (como PostgreSQL ou MySQL) está no modelo de armazenamento:

Característica	Armazenamento em Linhas	Armazenamento Colunar
Organização	Dados armazenados linha a linha	Dados armazenados coluna a coluna
Leitura	Lê todas as colunas da linha	Lê apenas as colunas necessárias
Compressão	Moderada (dados heterogêneos)	Alta (dados homogêneos por coluna)
Melhor para	OLTP (transações)	OLAP (análises)
Exemplo	PostgreSQL, MySQL	ClickHouse, Apache Druid

Em cenários analíticos típicos, onde as consultas acessam poucas colunas de tabelas com milhões ou bilhões de linhas, o armazenamento colunar é significativamente mais eficiente.

Arquitetura do ClickHouse

MergeTree Engine

O **MergeTree** é o engine de armazenamento principal do ClickHouse. Ele organiza os dados em *parts* que são periodicamente mescladas (*merged*) em segundo plano, otimizando a compressão e a performance de leitura.

Variantes do MergeTree incluem:

- **ReplacingMergeTree:** remove linhas duplicadas com base em uma chave de ordenação
- **SummingMergeTree:** agrega automaticamente valores numéricos ao mesclar
- **AggregatingMergeTree:** mantém estados de agregação pré-computados

© Copyright Tecnisys 2026. All rights reserved.

- **CollapsingMergeTree:** suporta remoção lógica de registros por colapso

Replicação e Sharding

O ClickHouse suporta replicação e sharding nativos para alta disponibilidade e escalabilidade:

- **Replicação:** utiliza o ZooKeeper (ou ClickHouse Keeper) para coordenar réplicas, garantindo consistência eventual dos dados
- **Sharding:** distribui os dados entre múltiplos nós utilizando chaves de distribuição configuráveis

No TDP Kubernetes, o ClickHouse é implantado com replicação habilitada por padrão, garantindo alta disponibilidade dos dados analíticos.

Casos de Uso no TDP

Análise de Logs e Métricas

O ClickHouse é ideal para armazenar e consultar grandes volumes de logs e métricas operacionais, oferecendo:

- Ingestão em tempo real de eventos
- Retenção configurável por TTL (Time-To-Live)
- Consultas ad-hoc com latência de milissegundos

Dashboards em Tempo Real

Integrado ao **Apache Superset**, o ClickHouse fornece a camada de dados para dashboards interativos com atualização em tempo real, especialmente útil para:

- Monitoramento de pipelines de dados
- KPIs de negócios em tempo real
- Análise de séries temporais

Data Warehousing

O ClickHouse pode ser utilizado como data warehouse para consultas analíticas complexas, complementando o **Trino** para cenários onde a latência de resposta é crítica.

SQL no ClickHouse

© Copyright Tecnisys 2026. All rights reserved.



O ClickHouse suporta uma sintaxe SQL rica e familiar, incluindo:

```
-- Criar tabela com engine MergeTree
CREATE TABLE eventos (
    data DateTime,
    usuario_id UInt64,
    evento String,
    valor Float64
) ENGINE = MergeTree()
ORDER BY (data, usuario_id);

-- Inserir dados
INSERT INTO eventos VALUES
    ('2024-01-01 10:00:00', 1001, 'login', 0),
    ('2024-01-01 10:05:00', 1001, 'compra', 150.00);

-- Consulta analítica com agregação
SELECT
    toDate(data) AS dia,
    count() AS total_eventos,
    countIf(evento = 'compra') AS total_compras,
    sum(valor) AS valor_total
FROM eventos
WHERE data >= '2024-01-01'
GROUP BY dia
ORDER BY dia;
```

Funções Especializadas

O ClickHouse oferece uma ampla biblioteca de funções otimizadas:

- **Funções de data/hora:** `toStartOfHour()`, `toMonday()`, `dateDiff()`
- **Funções de agregação:** `quantile()`, `uniq()`, `groupArray()`
- **Funções de arrays:** `arrayJoin()`, `arrayMap()`, `arrayFilter()`
- **Funções aproximadas:** `uniqHLL12()`, `quantileTDigest()` para estimativas rápidas

Integração com o Ecossistema TDP

No TDP Kubernetes, o ClickHouse se integra com os seguintes componentes:

© Copyright Tecnisys 2026. All rights reserved.

Componente	Integração
Apache Superset	Fonte de dados para dashboards e visualizações
CloudBeaver	Administração e consultas SQL via interface web
Apache NiFi	Ingestão de dados de diversas fontes para o ClickHouse
Trino	Consultas federadas envolvendo dados no ClickHouse
Apache Airflow	Orquestração de pipelines de carga e transformação

Boas Práticas

- **Escolha a chave de ordenação com cuidado:** a `ORDER BY` do MergeTree determina como os dados são organizados fisicamente; escolha colunas que serão frequentemente filtradas
- **Use tipos de dados apropriados:** prefira `UInt32` a `Int64` quando possível; use `LowCardinality(String)` para colunas com poucos valores distintos
- **Configure TTL para retenção:** utilize `TTL data + INTERVAL 90 DAY` para remover dados antigos automaticamente
- **Evite `SELECT *`:** consulte apenas as colunas necessárias para aproveitar a vantagem do armazenamento colunar
- **Utilize vistas materializadas:** para pré-computar agregações frequentes e acelerar dashboards

Detalhes do Projeto ClickHouse

O ClickHouse foi desenvolvido predominantemente em **C++**, aproveitando a performance nativa da linguagem para operações de baixo nível e processamento vetorizado de dados colunares.

© Copyright Tecnisys 2026. All rights reserved.

Recursos do ClickHouse

Fonte(s):

- clickhouse.com
- GitHub - ClickHouse
- ClickHouse SQL Reference

© Copyright Tecnisys 2026. All rights reserved.

CloudBeaver

Gerenciamento de Bancos de Dados via Web



> **This component is available only for TDP Kubernetes from v3.0.**

O **CloudBeaver** é uma ferramenta web open source de gerenciamento e administração de bancos de dados, desenvolvida pela DBeaver. Baseada na arquitetura do DBeaver (um dos clientes de banco de dados mais populares do mundo), o CloudBeaver oferece uma interface gráfica acessível via navegador, eliminando a necessidade de instalação de software cliente em cada estação de trabalho.

No TDP Kubernetes, o CloudBeaver é um componente **exclusivo** da edição Kubernetes, fornecendo uma interface web unificada para administração de todos os bancos de dados e engines de consulta da plataforma.

Por que CloudBeaver?

A inclusão do CloudBeaver no TDP Kubernetes traz diversos benefícios:

- **Acesso centralizado:** administre todos os bancos de dados do TDP a partir de um único ponto de acesso via navegador
- **Zero instalação no cliente:** não requer instalação de software nas estações de trabalho dos usuários
- **Interface intuitiva:** editor SQL com autocomplete, visualização de dados em tabela, e navegação por esquemas
- **Multi-banco:** suporta conexão simultânea com PostgreSQL, ClickHouse, Trino e outros engines SQL
- **Controle de acesso:** gerenciamento de usuários e permissões por conexão de banco de dados
- **Compartilhamento de conexões:** administradores podem configurar conexões compartilhadas para toda a equipe

© Copyright Tecnisys 2026. All rights reserved.

Principais Recursos

Editor SQL

O CloudBeaver disponibiliza um editor SQL completo com:

- **Autocomplete inteligente:** sugestões de tabelas, colunas e funções com base no contexto
- **Syntax highlighting:** destaque de sintaxe SQL para facilitar a leitura e escrita de queries
- **Múltiplas abas:** execução simultânea de consultas em diferentes bancos de dados
- **Histórico de consultas:** registro das consultas executadas para referência futura
- **Exportação de resultados:** exportação em CSV, JSON e outros formatos

Navegação de Esquemas

A interface permite navegar visualmente pela estrutura dos bancos de dados:

- Exploração de schemas, tabelas, colunas e índices
- Visualização de tipos de dados e constraints
- Inspeção de views e stored procedures
- Visualização de DDL (Data Definition Language) dos objetos

Visualização de Dados

O CloudBeaver oferece modos de visualização flexíveis para os resultados das consultas:

- **Modo tabela:** visualização tradicional em grid com ordenação e filtros
- **Modo texto:** visualização em formato texto para dados complexos
- **Edição inline:** edição direta dos dados na tabela (quando permitido)

Gerenciamento de Conexões

Os administradores podem gerenciar conexões de forma centralizada:

- Configurar conexões para todos os bancos de dados do TDP
- Definir quais conexões são compartilhadas com quais usuários
- Configurar templates de conexão para facilitar o acesso
- Monitorar conexões ativas e sessões de usuários

Integração com o TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.



No TDP Kubernetes, o CloudBeaver é pré-configurado para se conectar aos seguintes componentes:

Componente	Driver	Porta Padrão	Uso Típico
PostgreSQL	PostgreSQL JDBC	5432	Administração do banco de dados relacional
ClickHouse	ClickHouse JDBC	8123	Consultas analíticas e exploração de dados
Trino	Trino JDBC	8080	Consultas federadas em múltiplas fontes
Hive Metastore	Hive JDBC	10000	Exploração de metadados de tabelas

Fluxo de Trabalho Típico

1. O administrador configura as conexões dos bancos de dados do TDP no CloudBeaver
2. Os analistas acessam o CloudBeaver via navegador e se autenticam
3. As conexões compartilhadas aparecem automaticamente na lista de conexões disponíveis
4. Os analistas podem explorar esquemas, executar consultas SQL e exportar resultados
5. O histórico de consultas permite rastrear e reutilizar queries anteriores

Arquitetura

O CloudBeaver é composto por:

- **Frontend:** aplicação web React acessível via navegador
- **Backend:** servidor Java baseado no Eclipse Equinox (OSGi)

© Copyright Tecnisys 2026. All rights reserved.



- **Drivers JDBC:** conectores para cada tipo de banco de dados suportado
- **Workspace:** armazenamento de configurações, conexões e preferências dos usuários

No TDP Kubernetes, o CloudBeaver é implantado como um Deployment com um Service do tipo ClusterIP, acessível via Ingress Controller.

Segurança

O CloudBeaver oferece mecanismos de segurança para proteger o acesso aos dados:

- **Autenticação local:** gerenciamento de usuários e senhas pelo próprio CloudBeaver
- **Autenticação LDAP:** integração com diretórios LDAP/Active Directory para autenticação corporativa
- **Permissões por conexão:** controle granular de quais usuários podem acessar quais bancos de dados
- **Conexões somente leitura:** possibilidade de configurar conexões que permitem apenas consultas (SELECT)
- **Logs de auditoria:** registro de todas as ações realizadas pelos usuários

> **INFO**
>
> No TDP Kubernetes, recomenda-se configurar o CloudBeaver com autenticação LDAP quando disponível, alinhando-se com a estratégia de autenticação centralizada da organização.

Comparação com DBeaver

Característica	DBeaver Desktop	CloudBeaver
Instalação	Local (por estação)	Centralizada (servidor)
Acesso	Aplicação desktop	Navegador web
Gerenciamento	Individual	Centralizado por admin
Compartilhamento	Manual	Conexões compartilhadas

© Copyright Tecnisys 2026. All rights reserved.

Característica	DBeaver Desktop	CloudBeaver
Recursos	Completo	Subconjunto focado em consultas
Ideal para	Desenvolvimento	Equipes e ambientes cloud

Boas Práticas

- **Configure conexões somente leitura** para analistas que não precisam modificar dados
- **Organize conexões em pastas** agrupando por ambiente (desenvolvimento, produção) ou por componente
- **Utilize templates de conexão** para padronizar configurações entre a equipe
- **Monitore sessões ativas** para identificar consultas de longa duração que possam impactar a performance
- **Habilite autenticação LDAP** em ambientes corporativos para evitar proliferação de credenciais

Detalhes do Projeto CloudBeaver

O CloudBeaver tem seu Frontend desenvolvido em **TypeScript/React** e o Backend em **Java**, baseado no framework Eclipse Equinox (OSGi), compartilhando a mesma base do DBeaver Desktop.

Recursos do CloudBeaver

Fonte(s):

- dbeaver.com/cloudbeaver
- GitHub - CloudBeaver
- CloudBeaver Wiki

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake



Estrutura de Armazenamento

Em arquiteturas modernas de dados, a **camada de armazenamento** é a base que permite guardar grandes volumes de dados de forma escalável e econômica. Ela utiliza sistemas como Amazon S3, Azure Data Lake Storage ou HDFS para armazenar arquivos em formatos como Parquet, ORC ou Avro. Essa camada, no entanto, não impõe estrutura, controle de versão, validação de esquema nem transações — ela apenas armazena os dados.

Sobre essa base surgiu o conceito de **Data Lake**: uma arquitetura voltada para armazenar dados brutos em larga escala, oriundos de diversas fontes, em diferentes formatos. A promessa dos data lakes era centralizar tudo em um só lugar — facilitando análises futuras, integrando dados variados e reduzindo custos. Isso os tornou populares em projetos de ciência de dados e pipelines de ingestão massiva.

Por que os Data Lakes se popularizaram

Mesmo com limitações técnicas, os data lakes se popularizaram por sua flexibilidade e economia. Eles permitem:

- **Armazenar dados em seu formato original**, sem necessidade de modelagem antecipada.
- **Coletar dados em lote e em tempo real**, com capacidade quase ilimitada de expansão.
- **Atender múltiplos perfis de usuários e ferramentas**, favorecendo análise ad hoc, machine learning, relatórios e dashboards.
- **Consolidar e democratizar o acesso aos dados**, rompendo silos de sistemas legados.

© Copyright Tecnisys 2026. All rights reserved.

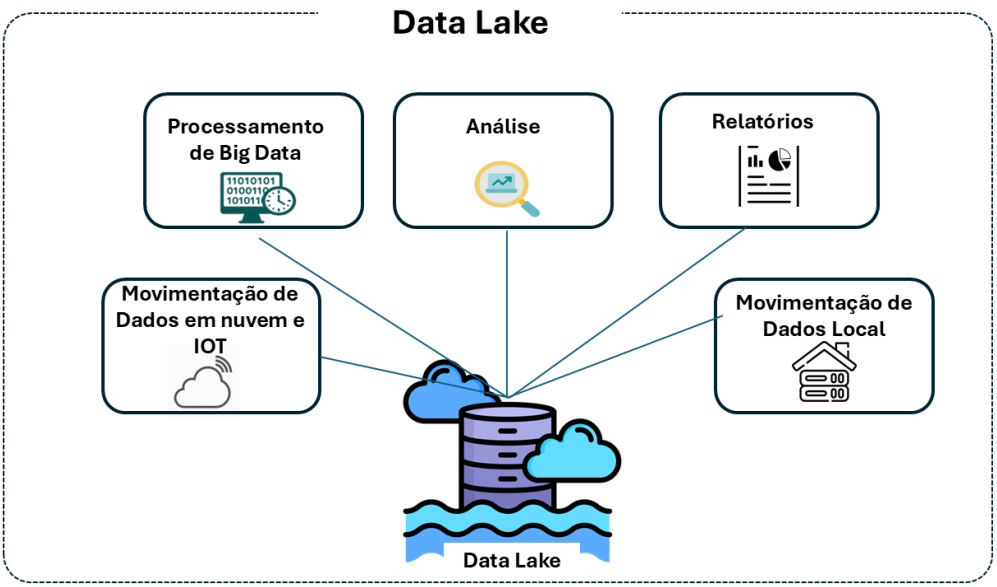


Figura 1 - Data Lake

Limitações dos Data Lakes

Apesar dos benefícios, os data lakes tradicionais não oferecem os controles necessários para ambientes analíticos confiáveis:

- **Não garantem transações ACID**: múltiplas gravações podem causar inconsistência ou corrupção.
- **Não possuem controle de versão dos dados**: não é possível consultar ou restaurar o estado anterior de uma tabela.
- **Não validam a consistência do esquema**: erros de estrutura podem passar despercebidos, comprometendo as análises.
- **Não oferecem performance otimizada para consultas**: arquivos pequenos e desorganizados afetam o desempenho.

Essas limitações impedem o uso do data lake como base confiável para aplicações analíticas críticas. Para superar esses problemas, foi criada uma nova camada — o **Delta Lake**.

O que é o Delta Lake

O **Delta Lake** é uma camada transacional que opera sobre um data lake existente. Ele não substitui o data lake — ele o complementa. Adicionando transações ACID, controle de versão, gerenciamento de esquema e integração com ferramentas como Apache Spark, o Delta Lake transforma o data lake bruto em uma plataforma mais confiável, robusta e auditável — conhecida como **Lakehouse**[1].

© Copyright Tecnisys 2026. All rights reserved.

Características do Delta Lake

O Delta Lake introduz funcionalidades essenciais que corrigem as deficiências do data lake tradicional e viabilizam uma arquitetura de dados confiável e escalável. Entre seus principais recursos:

- **Transações ACID**: garantem atomicidade, consistência, isolamento e durabilidade mesmo em ambientes distribuídos. Leituras e gravações são seguras e previsíveis.
- **Controle de versão e time travel**: todas as alterações nos dados são versionadas automaticamente, permitindo consultar o estado de uma tabela em qualquer ponto no tempo, reverter alterações ou realizar auditorias históricas.
- **Gerenciamento e validação de esquema**: evita a ingestão de dados com estruturas incorretas. O esquema pode evoluir com controle, sem comprometer a integridade das tabelas.
- **Unificação de lote e streaming**: a mesma tabela Delta pode receber dados em lote e por streaming, oferecendo consistência entre diferentes modos de ingestão.
- **Desempenho otimizado para leitura**: inclui técnicas como compactação de pequenos arquivos, ordenação por colunas (Z-order) e *data skipping*, que aceleram drasticamente as consultas analíticas.
- **Integração com o ecossistema Spark**: permite executar operações SQL, APIs estruturadas e processamento em tempo real diretamente sobre as tabelas Delta.

Arquitetura do Delta Lake

A arquitetura do Delta Lake se apoia sobre o data lake tradicional, introduzindo uma camada transacional que garante confiabilidade e desempenho. Essa camada é composta por três elementos fundamentais:

- **Camada de Armazenamento**: Utiliza o mesmo repositório do data lake, como Amazon S3, Azure Data Lake Storage ou HDFS, onde os dados são armazenados em formato Parquet. Essa camada física continua a oferecer escalabilidade e economia.
- **Delta Log**: Um log de transações que registra todas as alterações efetuadas nas tabelas. Esse log possibilita o controle de versão, garantindo que cada modificação seja rastreada e permitindo funcionalidades de time travel e auditoria.

© Copyright Tecnisys 2026. All rights reserved.

- **Tabelas Delta**: São a abstração lógica que une os dados armazenados e o Delta Log. Através dessa integração, as tabelas Delta oferecem transações ACID, gerenciamento e evolução de esquema, além da unificação de ingestão batch e streaming.

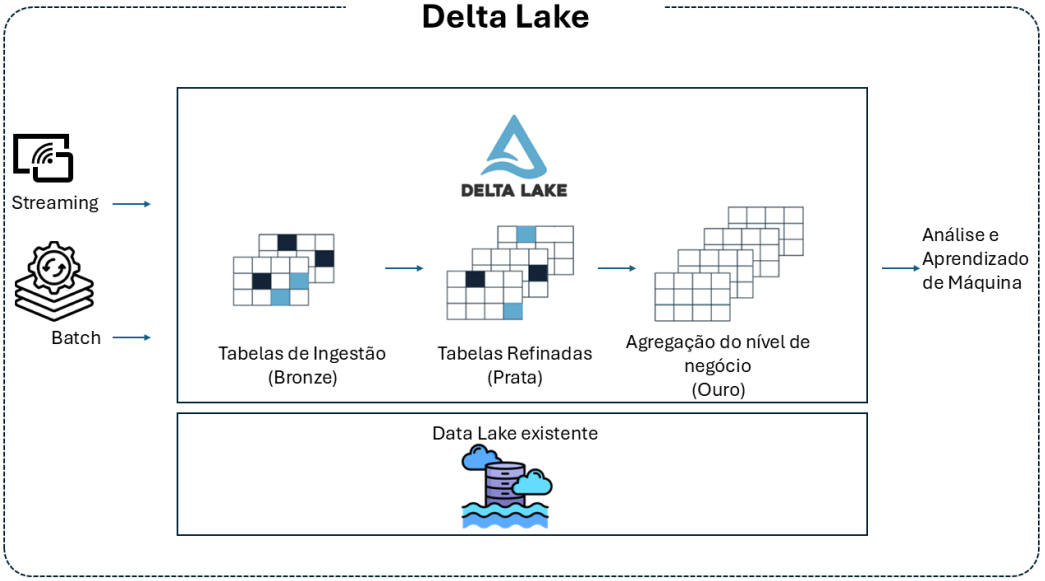


Figura 2 - Arquitetura do Delta Lake

Como o Delta Lake funciona

O Delta Lake opera continuamente sobre o armazenamento físico, gerenciando alterações nos dados e garantindo transações seguras e rastreáveis. Seu funcionamento se organiza em três áreas principais:

Ingestão de Dados

O Delta Lake suporta múltiplas formas de ingestão:

- **Streaming**: Integra-se ao Apache Spark Structured Streaming para ingestão contínua de dados, permitindo atualização em tempo real com consistência garantida (*exactly-once semantics*).

- **Batch**: Permite ingestões em lote em grande volume, com suporte a operações como `MERGE`, `UPDATE`, `DELETE` e `INSERT`, sem comprometer a integridade das tabelas.

Processamento de Consultas

As tabelas Delta podem ser consultadas utilizando:

© Copyright Tecnisys 2026. All rights reserved.



- **SQL**: Consultas interativas ou analíticas podem ser feitas diretamente sobre os dados versionados.
- **APIs de alto nível**: O Delta Lake é compatível com as APIs do Apache Spark (DataFrame, SQL, Structured Streaming).
- **Time travel**: Através da manipulação do Delta Log, é possível consultar o estado dos dados em qualquer ponto no tempo, de forma nativa.

Gestão de Dados

O Delta Lake oferece mecanismos para otimização contínua:

- **Compactação de arquivos pequenos**: Operações como `OPTIMIZE` reorganizam os dados para melhorar o desempenho das consultas.
- **Ordenação por Z-order**: Organiza os dados em colunas com alta seletividade, reduzindo o custo de leitura.
- **Limpeza de dados obsoletos**: Com o comando `VACUUM`, arquivos não referenciados são removidos com segurança, liberando espaço e mantendo a performance.

Esses recursos operam de forma integrada, mantendo a integridade dos dados mesmo em cenários com alta concorrência ou ingestão contínua.

Recursos do Delta Lake

- **Transações ACID no Spark**: níveis de isolamento serializáveis garantem que os leitores nunca vejam dados inconsistentes.
- **Manipulação escalável de metadados**: aproveita o poder de processamento distribuído do Spark para manipular todos os metadados para tabelas em escala de petabytes com bilhões de arquivos.
- **Unificação de streaming e lote**: uma tabela no Delta Lake é uma tabela de lote, além de fonte e coletor de streaming. A ingestão de dados de streaming, preenchimento de histórico em lote e consultas interativas funcionam com perfeição.
- **Aplicação de esquema**: manipula automaticamente variações de esquema para evitar a inserção de registros inválidos durante a ingestão.
- **Time travel**: o controle de versão de dados permite reversões, trilhas de auditoria históricas completas e experimentos de aprendizado de máquina reproduzíveis.
- **Upserts e exclusões**: suporta operações de mesclagem, atualização e exclusão para permitir casos de uso complexos, como captura de dados alterados, operações combinado dimensão de alteração lenta (SCD), upserts de streaming, etc.

© Copyright Tecnisys 2026. All rights reserved.



- **Ecossistema de conectores**: possui conectores que leem e gravam tabelas Delta de vários mecanismos de processamento de dados, como Apache Spark, Apache Flink, Apache Hive, Trino, AWS Athena, etc.

Quando usar o Delta Lake

Delta Lakes são uma ótima tecnologia de armazenamento de dados flexível e de baixo custo. Podem ser uma ótima opção para cenários que requerem:

- **Dados em vários formatos** provenientes de várias fontes;
- **Uso dos dados em muitas tarefas posteriores distintas**, por exemplo, análise, ciência de dados, aprendizado de máquina, etc.;
- **Flexibilidade** para executas muitos tipos distintos de consultas sem a obrigação de fazer perguntas com antecedência;
- **Processamento de dados em tempo real**: ingestão e análise de dados em tempo real com garantia de consistência.
- **Gerenciamento de grandes volumes de dados**: suporte a petabytes de dados com desempenho otimizado.
- **Necessidade de auditoria e versionamento**: acesso a versões anteriores dos dados para auditoria e recuperação.

Quando o Delta Lake não se aplica

O Delta Lake pode não ser a melhor escolha para:

- **Sistemas OLTP de alta concorrência**: aplicações que requerem transações de baixa latência e alta concorrência podem se beneficiar mais de bancos de dados relacionais tradicionais.
- **Necessidade de suporte a constraints complexas**: como chaves estrangeiras e triggers, que não são suportadas nativamente pelo Delta Lake.

Interação com Apache Spark

O Delta Lake é totalmente compatível com as APIs do Apache Spark e foi desenvolvido para integração completa com o Structured Streaming, permitindo o uso fácil de uma única cópia de dados para operações em lote e streaming, além de fornecer processamento incremental em escala. Permite:

- **Ingestão e processamento de dados em tempo real**: utilizando Spark Structured Streaming.

© Copyright Tecnisys 2026. All rights reserved.



- **Execução de consultas SQL e operações DML**: com suporte a transações ACID.
- **Otimizações de desempenho**: aproveitando as capacidades de processamento distribuído do Spark.

Diferenças entre Delta Lake e SGBDs tradicionais

- **Armazenamento**: o Delta Lake utiliza arquivos Parquet em sistemas de arquivos distribuídos, enquanto SGBDs tradicionais utilizam armazenamento em blocos.
- **Transações**: o Delta Lake oferece transações ACID em ambientes distribuídos, enquanto SGBDs tradicionais gerenciam transações em sistemas centralizados.
- **Evolução de esquema**: o Delta Lake suporta evolução de esquema com flexibilidade, enquanto SGBDs tradicionais requerem alterações estruturais mais rígidas.

Boas Práticas para Delta Lake

- **Compactação de arquivos pequenos**: utilizar o comando OPTIMIZE para melhorar o desempenho das consultas.
- **Ordenação por Z-order**: aplicar Z-ordering em colunas frequentemente utilizadas em filtros para acelerar as consultas.
- **Gerenciamento de versões**: utilizar o time travel para auditorias e recuperação de dados.

Detalhes do Projeto Delta Lake

- **Linguagem de Programação**: desenvolvido em Scala e Java.
- **Licença**: open source sob a licença Apache 2.0.
- **Integração**: compatível com Apache Spark, Apache Flink, Presto, Trino, entre outros.

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **4.0.1**, distribuída pelo Helm Chart `tdp-deltalake` v3.0.1.

© Copyright Tecnisys 2026. All rights reserved.



> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- Delta Lake Documentation
- GitHub - Delta Lake

Footnotes

1. A arquitetura lakehouse combina as vantagens dos data lakes e dos data warehouses em uma única plataforma. Usa o armazenamento barato e escalável de um Data Lake, oferece transações ACID, controle de versão, esquema e catálogos como um Data Warehouse, suporta processamento batch e streaming no mesmo local e permite SQL e machine learning sobre os mesmos dados. ↩

© Copyright Tecnisys 2026. All rights reserved.

Apache Druid

Database Analítico Real-Time



A habilidade de armazenar, processar e recuperar dados em velocidades muito rápidas é um dos principais requisitos atendidos por um *Database* analítico *Real-Time*.

Nos SGBDs tradicionais, derivar valor dos dados é uma tarefa difícil porque os dados precisam ser movidos, transformados, carregados em um *Datawarehouse* ou *Data Lake* antes de seu consumo. Isto demanda tempo, algumas horas, muitas vezes dias.

Mecanismos de *streaming*, como o Kafka, auxiliam, mas somente até certo ponto, pois a necessidade fundamental é um banco de dados com poder de processar grandes quantidades de dados em tempo real.

Os bancos de dados analíticos em tempo real otimizam recursos para permitir pesadas cargas de trabalho. Contam com protocolos de ingestão leves e estruturas de armazenamento eficientes em disco, para permitir ingestões muito rápidas. Sua arquitetura utiliza processamento massivamente paralelo com alto grau de simultaneidade, de forma a não resultar em altos custos com infraestrutura.

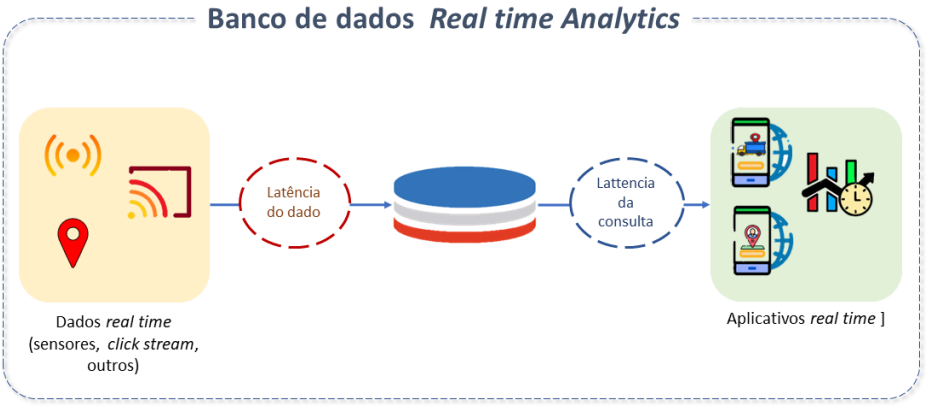


Figura 1 - Aspectos da Governança de dados

Características do Apache Druid

O Apache Druid é um Sistema Gerenciador de Banco de dados analítico *real-time*, projetado para análises rápidas e fragmentadas (consultas OLAP) em grandes conjuntos

© Copyright Tecnisys 2026. All rights reserved.

de dados. Potencializa *Use Cases* onde a ingestão *real-time*, o desempenho rápido de consultas e a alta produtividade são importantes.

Sua origem remonta a 2011, quando a equipe de dados de uma empresa de TI decidiu criar seu próprio banco de dados, após tentar o uso de várias alternativas de mercado para solucionar um problema de agregação e consulta rápida de dados em tempo real provenientes da Internet para analisar leilões de publicidade digital. A primeira versão do Druid escaneou, filtrou e agregou um bilhão de linhas em 950 milissegundos.

Druid tornou-se código aberto após alguns anos, e um projeto de alto nível da Apache Software Foundation em 2016.

Em 2023, mais de 1400 organizações usam o Druid e a ferramenta conta com mais de 10.000 desenvolvedores ativos em sua comunidade.

Dentre suas principais características, destacamos:

- **Escalabilidade e Flexibilidade:** Sua arquitetura elástica e distribuída permite a criação de qualquer aplicativo em qualquer escala.

- **Eficiência e Integração:** O lema do Druid é "faça apenas se for necessário", o que minimiza o trabalho do *Cluster*:

    - Não carrega dados do disco para a memória (ou vice-versa) quando não são necessários.
    - Não decodifica dados se puder operar diretamente em dados codificados.
    - Não lê o conjunto de dados completo se puder ler um índice menor.
    - Não inicia novos processos para cada consulta se puder usar um processo de execução longa.
    - Não envia dados desnecessários entre processos ou servidores.
    - O mecanismo de consulta e formato de armazenamento do Druid são totalmente integrados e foram projetados em conjunto para minimizar a quantidade de trabalho a ser realizado pelos servidores de dados.
    - É geralmente usado como um *backend* de Banco de Dados para GUI ("Graphical User Interface") de aplicações analíticas, ou para APIs altamente concorrentes que demandam rápidas agregações.
- **Resiliência e Durabilidade:** Druid é "auto-reparável", "autobalanceado" e tolerante a falhas. Foi projetado para funcionar continuamente sem inatividade, mesmo durante alterações de configuração e atualizações de software, impedindo a perda

© Copyright Tecnisys 2026. All rights reserved.

de dados, mesmo em caso de grandes falhas de sistemas. Seu *Cluster* se reequilibra automaticamente em segundo plano sem inatividade. Quando um servidor falha, o sistema "absorve" a falha e continua a operar.

- **Alta Performance:** Os recursos do Druid se combinam para permitir alto desempenho em alta simultaneidade, evitando trabalho desnecessário.

- **Alta Simultaneidade:** Este foi um dos objetivos originais do projeto Druid. Muitos *Clusters* suportam centenas de milhares de consultas por segundo. A chave disso é o relacionamento exclusivo entre armazenamento e recursos de computação. Os dados são armazenados em segmentos, que são verificados em paralelo por consultas de dispersão/reunião.

- **Ingestão de dados em alta velocidade:** Para ingestão *streaming* os gerentes intermediários e indexadores estão habilitados a responder a consultas em tempo real. Todas as tabelas estão sempre totalmente indexadas, tornando desnecessária a criação de índices.

  O Druid funciona melhor com dados orientados a eventos. As áreas de aplicação comuns para Druid incluem:

  - Análise de *Clickstream* (fluxos de cliques), incluindo análises web e mobiles.
  - Análise de telemetria de rede, incluindo monitoramento de desempenho de rede.
  - Armazenamento de métricas do servidor.
  - Análise da cadeia de suprimentos, incluindo métricas de "fabricação".
  - Métricas de desempenho de aplicativos.
  - Análise de marketing/publicidade digital.
  - Inteligência de negócios (OLAP).

© Copyright Tecnisys 2026. All rights reserved.

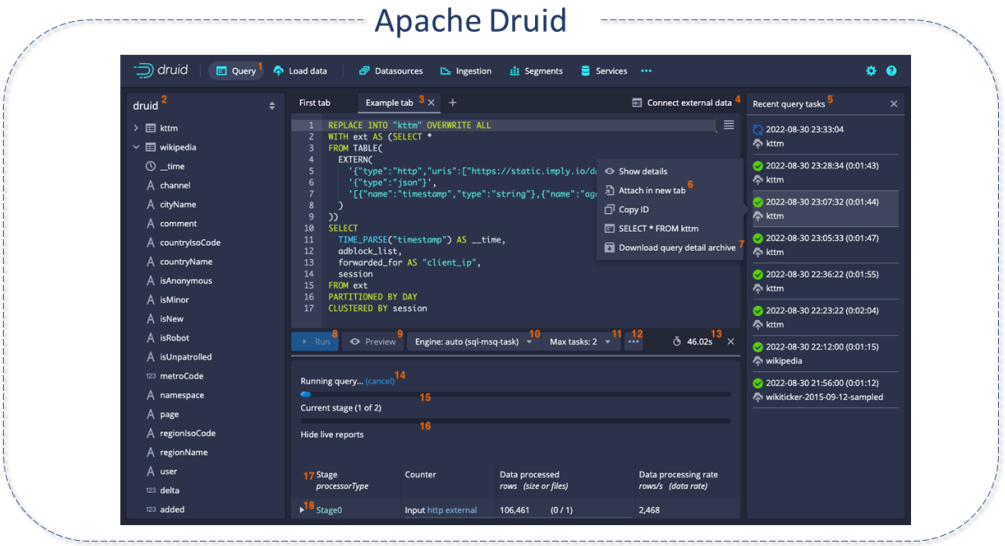


Figura 2 - Interface Apache Druid

Arquitetura do Apache Druid

Apache Druid tem uma arquitetura distribuída desenhada para ser *cloud friendly* e fácil de operar. Serviços podem ser configurados e escalados de forma independente, garantindo um máximo de flexibilidade nas operações de *Cluster*.

Druid combina ideias de data warehouses, bancos de dados de séries temporais e sistemas *logsearch* (busca de *log*).

Alguns dos seus principais componentes incluem:

Serviços Druid (Tipos de Processo)

- Coordenador: gerencia a disponibilidade dos dados no *Cluster*.
- *Overlord*: controla a designação de *workloads* (cargas de trabalho) para ingestão de dados.
- *Broker*: lida com consultas de clientes externos.
- Serviços do roteador: são opcionais. Encaminham solicitações para *Brokers*, *Coordenadores* e *Overlords*.
- Serviços Históricos: armazenam dados que podem ser consultados.
- Serviços de *MiddleManager*: ingerem os dados para a maior parte dos métodos de ingestão.
- Indexador: é um recurso opcional e experimental. Seu sistema de gerenciamento de memória ainda estava em desenvolvimento quando da confecção desta documentação. Será aprimorado em versões posteriores. Deverá atuar como uma

© Copyright Tecnisys 2026. All rights reserved.



alternativa ao *MiddleManager* + *Peon*. Em vez de "bifurcar" um processo JVM separado por tarefa, o *indexer* executa tarefas como encadeamentos separados dentro de um processo JVM único.

> **NOTE**
>
> Os serviços podem ser visualizados na Guia Serviços, no console da Web.

Servidores

Os servidores Druid podem ser implantados da maneira desejada. Para facilitar a implantação, sugere-se organizá-los em três tipos:

- Mestre: Executa os processos do Coordenador e do *Overlord*, gerencia a disponibilidade e ingestão de dados.
- Query: Executa processos de *Broker* e *Router* opcional e lida com consultas de clientes externos.
- Dados: executa processos históricos e do *MiddleManager*, executa cargas de trabalho de ingestão e armazena todos os dados que podem ser consultados.

Dependências Externas:

Além de seus tipos de processos integrados, o Druid também possui três dependências externas que alavancam a infraestrutura existente, quando presente:

- *Deep Storage*: Como parte da ingestão, o Druid armazena com segurança uma cópia do segmento de dados em *Deep Storage*, criando uma cópia adicional contínua e automatizada dos dados na nuvem ou HDFS. Disponibiliza o segmento para consultas imediatamente e cria uma réplica de cada segmento de dados. Sempre será possível recuperar dados do *Deep Storage*, mesmo no caso improvável de todos os servidores falharem.
- Armazenamento de Metadados: O armazenamento de metadados contém vários metadados de sistemas compartilhados, como informações de uso de segmentos e informações de tarefas. Em uma implantação em *Cluster*, geralmente é um RDBMS tradicional, como PostgreSQL. Em uma implantação de servidor único, pode ser um banco de dados Apache Derby local.
- *Zookeeper*: Usado para gerenciamento do *Cluster ativo*, coordenação e eleição de líder.

© Copyright Tecnisys 2026. All rights reserved.

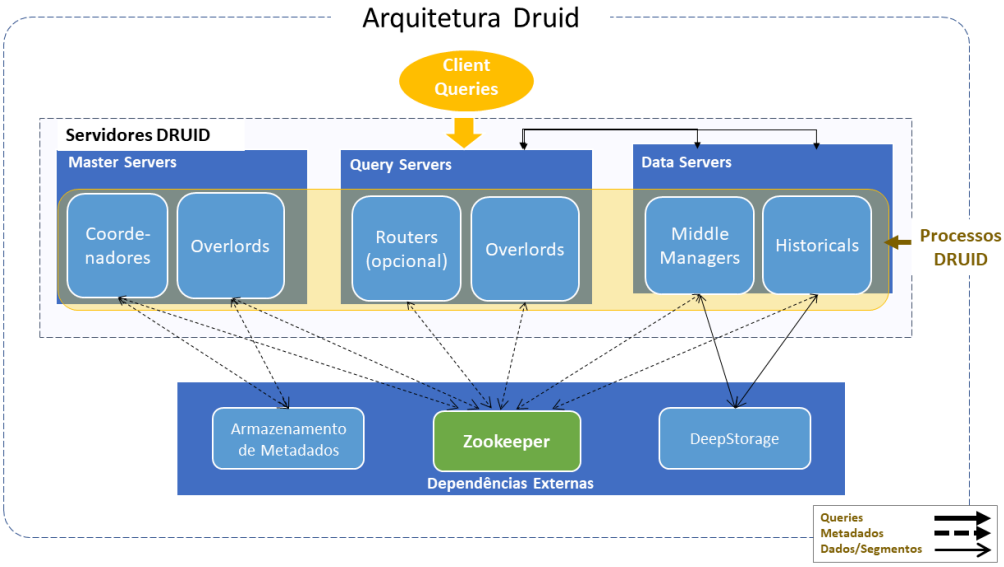


Figura 3 - Arquitetura Druid

Como o Apache Druid funciona

Design de Armazenamento

- ***Datasources* e Segmentos**:

  Os dados Druid são armazenados em *datasources*, que são similares a tabelas de um *RDBMS* tradicional. Cada *datasource* é particionado por tempo e, opcionalmente, por outros atributos. Cada intervalo de tempo é chamado *chunk* (bloco) - por exemplo, um único dia, no caso de particionado por dia. Dentro de um *chunk*, dado é particionado em um ou mais segmentos. Cada segmento é um arquivo único. Uma vez que os segmentos são organizados em blocos de tempo, algumas vezes é adequado pensar segmentos *vivendo* em uma linha do tempo.

  Uma fonte de dados pode ter de apenas alguns segmentos até centenas de milhares ou milhões de segmentos. Cada segmento é criado por um MiddleManager como *mutable* (mutável) e *uncommitted* (não confirmado). Os dados podem ser consultados assim que adicionados a um segmento *uncommitted*. O processo de construção do segmento acelera consultas posteriores, produzindo um arquivo de dados compacto e indexado:

  - Convertido para o formato colunar.
  - Indexado com índices de bitmap.
  - Compactados com reconhecimento de tipo para todas as colunas.

© Copyright Tecnisys 2026. All rights reserved.

Periodicamente, os segmentos são *committed* (confirmados) e publicados no *deep storage*, tornando-se imutáveis, e passados dos *MiddleManagers* para os processos históricos. Uma entrada sobre o segmento também é gravada no *Metadata Storage*. Esta entrada é um bit autodescritivo de metadados sobre o segmento, incluindo coisas como o esquema do segmento, seu tamanho e localização no *deep storage*, informando aos Coordenadores quais dados estão disponíveis no *Cluster*.

- **Indexação e Transferência**:

  Indexing é o processo pelo qual novos segmentos são criados. *Handoff* é o processo com o qual são publicados e começam a ser servidos por processos históricos.

- **Identificadores de Segmentos**:

  Todo segmento tem um identificador de 4 partes com os seguintes componentes:

  - Nome do *datasource*
  - Intervalo de tempo (para o bloco de tempo que contém o segmento), que corresponde ao *segmentGranularity* especificado no tempo de ingestão.
  - Número de versão (geralmente umISO8601 correspondente a quando o conjunto de segmentos foi iniciado). Serve para o controle da simultaneidade de várias versões.
  - Número da partição (um inteiro, único dentro do *datasource*+intervalo+versão, não necessariamente contíguo).

- **Ciclo de Vida dos Segmentos**:

  Cada segmento tem um ciclo de vida que envolve:

  - Armazenamento de Metadados: os metadados do segmento são armazenados no *Metadata Storage* assim que o segmento termina de ser construído (publicação).
  - Armazenamento profundo: os arquivos de dados do segmento são enviados para o *deep storage* assim que o segmento termina de ser construído, antes da publicação.
  - Disponibilidade para consulta: os segmentos estão disponíveis para consulta em um servidor de dados Druid, como uma tarefa em tempo real ou um processo histórico.

Ingestão:

© Copyright Tecnisys 2026. All rights reserved.

A carga de dados no Druid é denominada *ingestion* (ingestão) ou *indexing* (indexação). Quando o dado é ingerido no Druid, o Druid lê o dado do sistema de origem e armazena-o em arquivos de dados denominados *segments* (segmentos). Em geral, os segmentos contêm alguns milhões de linhas cada.

Para a maior parte dos métodos de ingestão, o processo *MiddleManager* ou o processo *Indexer* irá carregar os dados. A única exceção é para a ingestão baseada no *Hadoop*, que usa o *job* MapReduce no Yarn.

Durante a ingestão, o *Druid* cria segmentos e os armazena no *Deep storage*. Nós históricos carregam os segmentos na memória para responder às consultas. Para ingestões *streaming*, os *MiddleManagers* e os *Indexers* podem responder a consultas em tempo real quando o dado está sendo carregado.

Os métodos de ingestão mais comuns são:

- ***Streaming***: para o qual há duas opções: *Kafka* e *Kinesis*.
    - ***Kafka***:

      Quando habilitado o serviço *Kafka indexing*, supervisores podem ser configurados no *Overlord* para gerenciar a criação e o *ciclo de vida* das tarefas de indexação *Kafka*. As tarefas de indexação *Kafka* lêem eventos usando a partição própria do *Kafka* e o mecanismo *offset* para garantir a ingestão *de uma única vez*. O Supervisor acompanha o estado das tarefas de indexação para:

        - Coordenar *Handoffs* (transferências).
        - Gerenciar falhas.
        - Garantir que a escalabilidade e requisitos de replicação estão mantidos.
    - ***Kinesis***:

      Quando habilitado, o serviço indexador *Kinesis*, é possível configurar supervisores no Overlord para gerenciar a criação e o ciclo de vida das tarefas de indexação *Kinesis*. Estas tarefas lêem os eventos usando seu próprio mecanismo de *shard*(fragmento) e número de sequência para garantir a ingestão de uma só vez. O supervisor supervisionará o estado das tarefas de indexação para:

© Copyright Tecnisys 2026. All rights reserved.



    - Coordenar *Handoffs* (transferências).
    - Gerenciar falhas.
    - Garantir que a escalabilidade e requisitos de replicação estão mantidos.
- ***Batch*** (em Lote): para o qual há três opções: *Native batch*, SQL ou *Hadoop-based*.

    - **Ingestão em lote *Native Batch*:**

      Apache Druid suporta dois tipos de indexação *native batch*:

        - Indexação paralela: que permite a execução de múltiplas tarefas de indexação concorrentemente.
        - Indexação simples: que executa uma única tarefa a cada vez.
    - **Ingestão em lote baseada em Hadoop:**

      A ingestão em lote baseada no *Hadoop* é suportada via uma tarefa. Estas tarefas podem ser enviadas para uma instância em execução de um *Overlord*. Para comparar os tipos de ingestão (baseado em *Hadoop*, nativo e nativo *em lote*), consulte o site da comunidade aqui.

    - **Ingestão em lote baseada em SQL:**

      Druid suporta a ingestão em lote baseada em SQL usando a extensão *druid-multi-stage-query*, que adiciona um engine de consulta *multi-stage* para SQL, que viabiliza a execução de operações SQL *Insert* e *Replace* como tarefas *batch*. Como recurso experimental, a operação *Select* também pode ser usada.

Processamento de consultas

As consultas são distribuídas pelo *Cluster* Druid e gerenciadas por um *Broker*. Primeiro entram no *Broker*, que identifica os segmentos com dados que podem pertencer àquela consulta. A lista de segmentos é sempre "quebrada" por tempo e também por outros atributos, dependendo de como a fonte de dados é particionada. O *Broker* identifica quais Históricos e *MiddleManagers* estão atendendo os segmentos e distribui uma subconsulta reescrita para cada um desses processos. Estes processos executam cada subconsulta e retornam os resultados aos *Brokers*, que os mescla para obter a resposta final, retornando ao chamador original.

Gestão do dado

© Copyright Tecnisys 2026. All rights reserved.



As operações de gestão de dados envolvendo *replacing* (substituição) ou *deletion* (exclusão) de segmentos incluem:

- **Updates (Atualização)**
- **Deletion (Exclusão)**
- **Schema Changes (Mudanças de Esquema)** para dados novos e existentes
- **Compactação** e Automatic Compaction (Compactação e Compactação Automática)

Consultas SQL:

As consultas podem ser feitas por meio do Druid SQL (Druid traduz consultas SQL em sua linguagem de consulta nativa) ou pelo SQL nativo do Druid.

O Plano SQL acontece no *Broker*. Para configurar o Plano e consulta *JDBC* as propriedades de execução do *Broker* devem ser configuradas.

Recursos do Apache Druid:

- **Formato de armazenamento colunar:** Druid usa armazenamento orientado a colunas. Significa que carrega apenas as colunas que precisa para uma consulta particular. Isto melhora a velocidade das consultas. Além disso, para suportar *scans* e agregações rápidas, Druid otimiza o armazenamento de cada coluna segundo seu tipo de dado.
- **Formato de dados otimizado:** Os dados ingeridos são automaticamente colunarizados, indexados por hora, codificados por dicionário, indexados por um *bitmap* e compactados com reconhecimento de tipo.
- **Ingestão em tempo real ou lote:** O Druid pode ingerir dados em tempo real ou em lotes. Os dados ingeridos ficam imediatamente disponíveis para consulta.
- **Índices para filtragem rápida:** O Druid usa índices de *bitmap* compactados Roaring ou CONCISE para criar índices para filtragem e pesquisas rápidas em várias colunas.
- **Particionamento baseado em tempo:** O Druid primeiro particiona os dados por tempo. Opcionalmente, pode implementar particionamento adicional com base em outros campos. As consultas baseadas em tempo acessam apenas as partições que correspondem ao intervalo de tempo da consulta, o que melhora significativamente o desempenho.

© Copyright Tecnisys 2026. All rights reserved.



- **Algoritmos aproximados:** O Druid inclui algoritmos para *count-distinct* aproximado, classificação aproximada e cálculo de histogramas e percentis aproximados. Estes algoritmos oferecem uso de memória limitado e geralmente são substancialmente mais rápidos do que cálculos exatos. Para situações em que a precisão é mais importante que a velocidade, o Druid também oferece contagem exata e classificação exata.
- **Resumo automático no momento da ingestão:** O Druid opcionalmente oferece suporte ao resumo de dados em tempo de ingestão. Esse pré-resumo agrega particularmente seus dados, levando potencialmente a economias de custo significativos e aumentos de desempenho.
- ***Engine* interativa de Consultas:** Utiliza dispersão/reunião para consultas de alta velocidade com dados pré-carregados na memória ou armazenamento local, para evitar movimentação de dados e latência na rede.
- **Camadas e QoS:** Camadas configuráveis com qualidade de serviço que permitem a relação custo x desempenho ideal para cargas de trabalho mistas. Garantem prioridade e evitam a contenção de recursos.
- **Ingestão de *streams*:** Uma integração "connector free" com plataformas de *streaming* permite "query on arrival", alta escalabilidade, baixa latência e consistência garantida.
- **Confiabilidade ininterrupta:** Serviços de dados automáticos incluindo *Backup* contínuo, recuperação automatizada e replicação de vários nós garantem alta disponibilidade e "durabilidade".

Quando usar o Apache Druid

Druid é usado por muitas organizações de distintos tamanhos e para diferentes *use-cases*. Será uma boa escolha para as seguintes situações:

- Altas taxas de inserção e poucas atualizações.
- Predomínio de consultas e relatórios de agregação, como, por exemplo, *group by* (*agrupamentos*).
- Latências de consultas segmentadas de 100ms a alguns segundos.
- Dados com componentes de tempo (o Druid inclui otimizações e opções de design especificamente relacionadas ao tempo).
- Mesmo na existência de mais de uma tabela, cada consulta atinge apenas uma grande tabela distribuída. As consultas podem potencialmente atingir mais de uma tabela de "pesquisa" menor.

© Copyright Tecnisys 2026. All rights reserved.



- Existem colunas de dados com alta cardinalidade (URLs, IDs de usuário) e há necessidade de contagem e classificação rápidas sobre elas.
- Carregar dados *Kafka*, HDFS, arquivos simples ou armazenamento de objetos como o Amazon S3.

Quando o Apache Druid não se aplica

- Atualizações de baixa latência dos registros existentes usando uma chave primária. O Druid suporta inserções de *streaming*, mas não atualizações de *streaming*. É possível executar atualizações usando trabalhos em lote em segundo plano.
- Criação de um sistema de relatórios off-line em que a latência da consulta não é muito importante.
- Necessidade de grandes junções, o que significa a união de uma grande tabela de fatos a outra grande tabela de fatos.

Interação Apache Druid com Spark

Druid e Spark são soluções complementares, uma vez que Druid pode ser usado para acelerar consultas OLAP no Spark.

Spark é um framework geral de computação em *Cluster* inicialmente projetado em torno do conceito de Conjuntos de dados distribuídos resilientes (RDDs). Os RDDs permitem a reutilização de dados, mantendo resultados intermediários na memória e permitem que o Spark forneça cálculos rápidos para algoritmos iterativos. Isso é especialmente bom para certos fluxos de trabalho, como aprendizado de máquina (onde a mesma operação pode ser aplicada repetidamente até que um resultado convirja).

A generalidade do Spark o torna muito adequado como mecanismo para processar (limpar ou transformar) dados. Embora forneça capacidade de consultar dados por meio do Spark SQL, como o *Hadoop*, as latências de consulta não são especificamente direcionadas para serem iterativas (subsegundos).

O foco do Druid é em consultas de latência extremamente baixa e é ideal para alimentar aplicativos usados por milhares de usuários, e onde cada consulta deve retornar rápido o suficiente para que os usuários possam explorar interativamente os dados.

O Druid indexa totalmente todos os dados e pode atuar como uma camada intermediária entre o Spark e um aplicativo.

© Copyright Tecnisys 2026. All rights reserved.



Uma configuração típica é processar dados no Spark e carregar os dados processados no Druid para acesso mais rápido. Veja detalhes da interação Druid e Spark.

Principais diferenças entre Apache Druid e os SGBDs tradicionais

- **Esquemas:** Druid armazena dados em *datasources*, similarmente a tabelas em tradicionais *RDBMS*. Os seus modelos de dados guardam similaridades com modelos de dados relacionais e *timeseries* (de série temporal).
    - Seus esquemas devem sempre incluir um *timestamp* primário. Ele usa este campo para particionar e classificar os dados. Também usa para identificar e recuperar dados rapidamente dentro do intervalo de tempo das consultas. E para gestão de dados, como *dropar time chunks*, sobrescrever *time chunks* e regras de retenção baseadas em tempo.
- **Dimensões:** Dimensões são colunas que o Druid armazena "como estão". Podem ser usadas para qualquer finalidade. Por exemplo, agrupamento, filtro ou agregações de dimensões no momento da consulta.
    - Com o *rollup* desativado, Druid trata o conjunto de dimensões como um conjunto de colunas a serem ingeridas. As dimensões se comportam exatamente como qualquer banco de dados sem suporte a um recurso "Rollup".
- **Métricas:** São colunas que o Druid armazena de forma agregada. São mais úteis com o *Rollup* ativado. Com uma métrica especificada, uma função de agregação pode ser aplicada a cada linha durante a ingestão.

Boas Práticas para Apache Druid

- **Rollup:** pode acumular dados à medida em que ingere para minimizar o volume de dados brutos que necessitam ser armazenados. Isto é uma forma de sumarização ou pre-agregação.

- **Particionamento e *sorting***: adequados podem ter um impacto substancial no seu desempenho.

- **Sketches para colunas de alta cardinalidade**:. Quando lidando com colunas de alta cardinalidade, como *user-ID* ou outros identificadores únicos, considere o uso de *sketches* para uma análise aproximada antes de operar com os valores reais.

    - Quando o *sketch* é utilizado, o Druid não armazena o dado original bruto, mas um *sketch* (esboço) dele que pode alimentar um cálculo posterior no momento

© Copyright Tecnisys 2026. All rights reserved.

da consulta.
    - Casos de uso populares incluem cálculo de contagem distinta e quantil.
    - Em geral, o *sketch* atende a dois propósitos: melhorar o *rollup* e reduzir o consumo de memória durante a consulta.
- **Strings x dimensões numéricas:**. Para ingerir uma coluna com uma dimensão de tipo numérico é necessário especificar o tipo da coluna na seção *dimensions*. Se o tipo é omitido, Druid vai ingerir a coluna como o tipo *string* default.

- **Segmentos:**: Para operar adequadamente sob pesadas cargas de consulta, é importante que o tamanho de arquivo do segmento esteja dentro do intervalo recomendado de 300 a 700 MB. Se for maior que isto, considere mudar a granularidade do intervalo de tempo do segmento ou particionar seu dado e/ou ajustar o *targetRowsPerSegment* no *partitionsSpec*. Um bom ponto de partida para este parâmetro é 5 milhões de linhas.

Detalhes do Projeto Apache Druid

Apache Druid foi desenvolvido em Java.

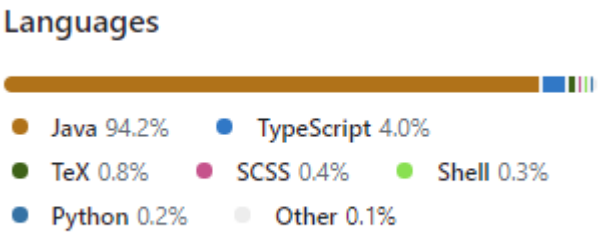


Figura 4 - Linguagens do Druid

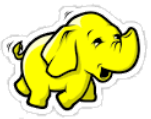
Fontes

- Apache Druid.org
- Apache Druid.org/technology
- GitHub - Apache Druid

© Copyright Tecnisys 2026. All rights reserved.

Apache Hadoop

Processamento distribuído de grande volume de dados



Apache Hadoop é uma implementação de código aberto do paradigma *MapReduce*, o qual define uma arquitetura para realização de processamento distribuído e paralelo de grandes volumes de dados, de modo que possam ser "executados" em vários servidores, usando modelos de programação simples.

> **NOTE**
>
> O paradigma *MapReduce* expande servidores únicos para muitas máquinas, oferecendo computação e armazenamento locais. A razão para sua escalabilidade é a natureza distribuída da sua solução - uma tarefa inicial complexa é dividida em várias menores, e, então, executada em máquinas diferentes. Posteriormente, o resultado é combinado para gerar a solução da tarefa inicial.

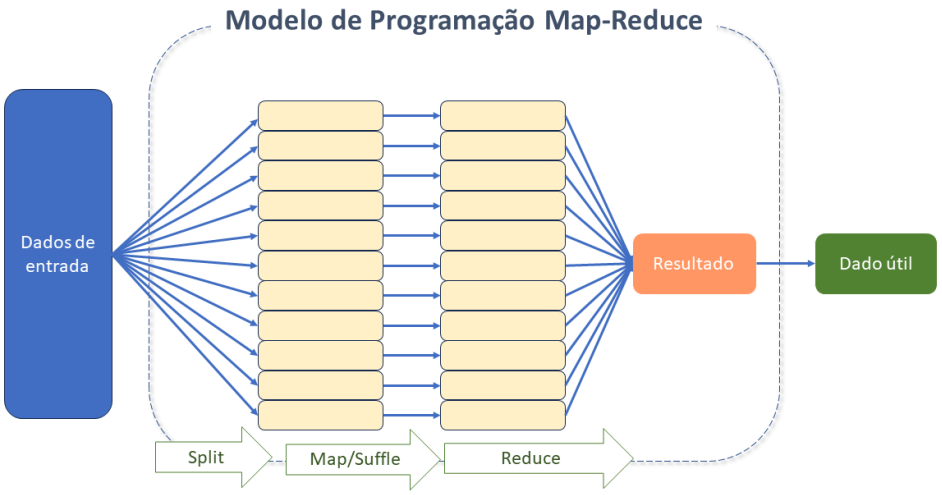


Figura 1 - Paradigma Map Reduce

© Copyright Tecnisys 2026. All rights reserved.



O Apache Hadoop surgiu em 2006, a partir de uma solução de busca na Web denominada Nutch Distributed File System (NDFS) que, segundo seus criadores, *Doug Cutting* e *Mike Cafarella*, teve sua gênese no documento *Google File System*, publicado em outubro de 2003. O NDFS evoluiu para o que hoje conhecemos como Haddop Distributed File System (HDFS).

O Apache Hadoop, bem como seu conjunto de soluções vinculadas, é uma das implementações mais populares e performáticas atualmente utilizadas para solucionar desafios relacionados à coleta, armazenamento e análise de dados, sejam estruturados ou não, estáticos ou tempo real.

Benefícios do Apache Hadoop

- **Escalabilidade:** Hadoop obtém escalabilidade de forma relativamente simples por meio de pequenas modificações em arquivos de configuração. O esforço para preparo de um ambiente com mil máquinas não é muito maior do que com 10. As limitações ficam por conta dos recursos, espaço em disco e capacidade de processamento disponíveis.
- **Simplicidade:** Hadoop gerencia questões relativas à computação paralela, como tolerância a falhas, escalonamento e balanceamento de carga. Suas operações se resumem a mapeamento (Map) e junção (Reduce), o que o mantém com foco na abstração e processamento de tarefas no modelo de programação MapReduce.
- **Baixo Custo/economia:**
  - Como software livre, dispensa aquisição de licenças e contratação de pessoal especializado.
  - Pode realizar processamento de dados em máquinas e redes convencionais.
  - Permite a execução de aplicações "na nuvem", sem a necessidade de implantar seu próprio aglomerado de máquinas.
- **Código Aberto:** Conta com uma comunidade ativa que envolve muitas empresas e programadores independentes compartilhando conhecimento, promovendo melhorias e colaborando com a sua qualidade. Este trabalho colaborativo promove atualizações e correções mais tempestivas e com maior qualidade, uma vez que a produção individual é avaliada por todos.
- **Alta disponibilidade:** O Hadoop não depende de hardware para entregar alta disponibilidade. Suas bibliotecas foram construídas para detectar e lidar com falhas na camada do aplicativo, fornecendo, assim, alta disponibilidade num *Cluster* de computadores.

© Copyright Tecnisys 2026. All rights reserved.

Arquitetura do Apache Hadoop

Os componentes-chave do Hadoop são o MapReduce e o HDFS. Com sua evolução, novos subprojetos foram incorporados, visando solucionar problemas específicos.

Principais componentes do Apache Hadoop

- **Hadoop Common:** É o "pilar" do Hadoop: onde fica armazenado um conjunto predefinido de utilitários e bibliotecas que podem ser usados por outros módulos dentro do Ecossistema Hadoop (arquivos e *scripts* Java Archive (JAR)).
- **HDFS:** É um Sistema de arquivos distribuídos, escalável, projetado para armazenar arquivos muito grandes e fornecer acesso de alta capacidade. Provê acesso com alto *throughput* (taxa de transferência) aos dados de aplicativos.
  - A ideia que norteia sua solução é garantir escalabilidade e confiabilidade ao processamento paralelo, utilizando o conceito de réplicas e blocos (com tamanho fixo de 64MB, em geral, armazenados em mais de um servidor).
- **Hadoop YARN:** Responsável pelo gerenciamento e agendamento dos recursos computacionais do *Cluster*.
- **Hadoop MapReduce:** É uma estrutura de software criada para processar grandes quantidades de dados. Uma excelente solução para processamento paralelo de dados.
  - Cada tarefa é especificada em termos de funções de mapeamento e redução. Ambas as tarefas rodam em paralelo, no *Cluster*, enquanto o sistema de armazenamento necessário é fornecido pelo HDFS.

Subprojetos do Apache Hadoop

A necessidade de novas alternativas para promover um processamento mais rápido e eficaz que os SGBD tradicionais aproximou grandes empresas da Plataforma, que, por se tratar de um projeto de código aberto, recebeu grandes estímulos.

Com o surgimento de novas necessidades, novas ferramentas foram criadas para serem executadas sobre o Hadoop. Conforme se consolidavam, passavam a integrá-lo. Foram essas contribuições que colaboraram com a rápida evolução do Apache Hadoop.

Abaixo as principais ferramentas do Ecossistema Hadoop, que compõem a Plataforma Tecnisys Data Platform (TDP):

© Copyright Tecnisys 2026. All rights reserved.



Figura 2 - Ecossistema Hadoop

Evolução do Apache Hadoop

> Veja aqui o Timeline - com a evolução do Apache Hadoop

Configurações e Uso

- Configurando um *Cluster* de nó único
- Configurando um *Cluster*
- Iniciando e Parando um *Cluster*
- Usando o File System shell
- Conhecendo os comandos e subprojetos do Hadoop

Diretrizes de Compatibilidade

© Copyright Tecnisys 2026. All rights reserved.

- Hadoop.apache.org -> Especificações destinadas à comunidade de desenvolvedores Hadoop
- Hadoop.apache.org -> Especificações destinadas a Administradores do Sistema

O Futuro do Apache Hadoop

Em abril de 2021, a Apache Software Foundation anunciou a retirada de 13 projetos relacionados a BigData, dez dos quais faziam parte do Ecossistema Hadoop (Eagle, Sentry, Tajo, etc.).

Esta ação levantou questões sobre o futuro do Apache Hadoop. Entretanto, a Plataforma possui um número representativo de grandes usuários, que continuam a utilizá-lo.

Segundo o Google Trends, o Apache Hadoop atingiu seu pico de popularidade entre 2014 e 2017 e, como em todo ciclo tecnológico, sua popularidade pode sim, entrar em declínio gradualmente em favor de outras ferramentas emergentes. Entretanto, a tecnologia continua progredindo, com foco na contínua evolução do HDFS, YARN e MapReduce. Sua última versão foi lançada em março de 2023.

Detalhes do Projeto Apache Hadoop

O Apache Hadoop foi desenvolvido em Java, predominantemente.

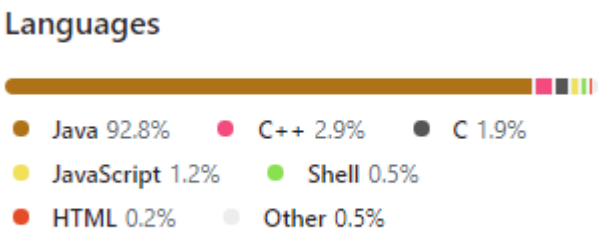


Figura 3 - Linguagens do Hadoop

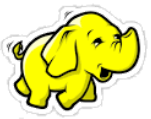
Fonte(s):

- https://hadoop.apache.org/
- https://cwiki.apache.org/confluence/display/hadoop
- https://github.com/apache/hadoop

© Copyright Tecnisys 2026. All rights reserved.

Apache HDFS

Sistema de Arquivos Distribuídos



Um sistema de arquivos distribuídos (DFS) é um sistema de arquivos que habilita o acesso, pelo cliente, a dados armazenados em múltiplos servidores, por meio de uma rede de computadores, como se estivesse acessando um armazenamento local.

Os arquivos são distribuídos em múltiplos servidores de armazenamento e em múltiplas localidades, o que permite o compartilhamento de dados e de recursos.

O DFS agrupa vários nós de armazenamento e neles distribui logicamente conjuntos de dados, cada um com sua própria configuração. Os dados podem residir em vários tipos de dispositivos, como unidades de estado sólido e discos rígidos.

Os dados são replicados, o que viabiliza a redundância para alcançar disponibilidade.

Para expandir a infraestrutura, a organização precisa apenas adicionar mais nós ao sistema.

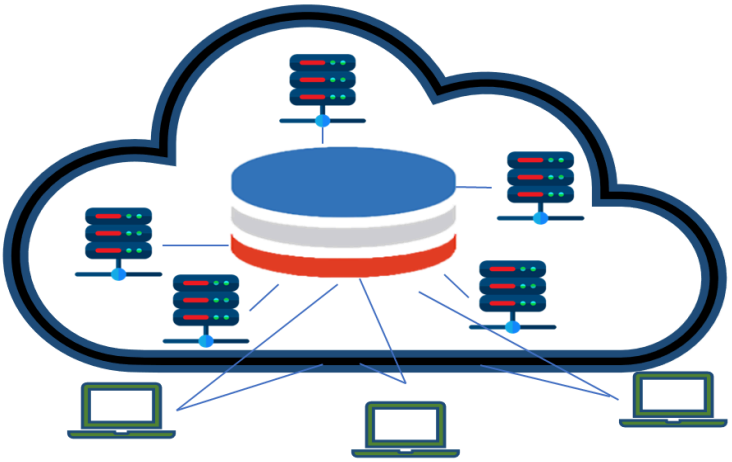


Figura 1 - DFS - Sistema de Arquivos Distribuídos

Características do Apache HDFS

© Copyright Tecnisys 2026. All rights reserved.

O HDFS (Hadoop Distributed File System) é o principal sistema de armazenamento utilizado pelo Hadoop.

Trata-se de um sistema de arquivos distribuídos extremamente confiável, desenhado para execução em hardware comum que trabalha com rápida transferência de dados entre nós.

O HDFS foi originalmente construído como infraestrutura para um projeto de mecanismo de pesquisa da *Web Apache Nutch* e hoje faz parte do *Core* do projeto *Apache Hadoop*. Possui muitas semelhanças com os sistemas de arquivo distribuídos existentes. Entretanto, suas diferenças são significativas:

- É altamente tolerante a falhas
- Foi projetado para hardware de baixo custo
- Oferece alta taxa de transferência no acesso aos dados
- É adequado para tratamento de grandes conjuntos de dados.

Destacamos abaixo algumas de suas principais características:

- *Tolerância a falhas*:
  Uma instância do HDFS pode consistir em milhares de servidores armazenando parte dos dados de um sistema de arquivos, o que envolve um grande número de componentes. Isto aumenta a probabilidade de falhas pois, em algum momento, um ou mais componentes poderá estar *não funcional*. Por esta razão a tolerância a falhas é a premissa principal do HDFS. A detecção de falhas e a recuperação rápida e automática é a sua meta arquitetônica central.

- *Acesso Streaming de dados*:
  O padrão de acesso aos dados é o *streaming* de dados, ou seja, dados gerados em tempo real e com fluxo contínuo.

- *Suporte a grandes conjuntos de dados*:
  Um arquivo comum no HDFS pode alcançar, tranquilamente, terabytes em tamanho, pois aplicativos que executam no HDFS tratam grandes conjuntos de dados. O HDFS foi ajustado para fornecer elevada *largura de banda agregada* , escalar para centenas de nós em um único *Cluster* e suportar dezenas de milhões de arquivos em uma única instância.

© Copyright Tecnisys 2026. All rights reserved.

- *Modelo de Coerência Simples*:
  O HDFS foi projetado com base no princípio *write once read many*(gravação única, múltiplas leituras). Depois que os dados são gravados, não há suporte para atualizações em um ponto arbitrário ou alterações exceto em *appends* e *truncates*(anexar o conteúdo ao final dos arquivos ou excluir todas as linhas). É essa premissa que simplifica os problemas de coerência de dados e garante alto *throughput* no acesso a dados. Um aplicativo *MapReduce* ou um *web crawler*(rastreador de sites) se encaixam perfeitamente neste modelo.

- *Interfaces que aproximam os aplicativos dos dados*:
  Uma computação solicitada por um aplicativo é sempre mais eficiente se executada próxima aos dados com os quais opera. Principalmente quando o tamanho do conjunto de dados é grande, pois minimiza o congestionamento da rede e aumenta a taxa de transferência geral do sistema. O HDFS oferece *interfaces* para que os aplicativos se aproximem dos dados.

- *Portabilidade entre plataformas heterogêneas de hardware e software*:
  O HDFS foi projetado para ser facilmente "portável" de uma plataforma para outra. Isto facilita a sua adoção generalizada como plataforma para grande conjunto de aplicativos.

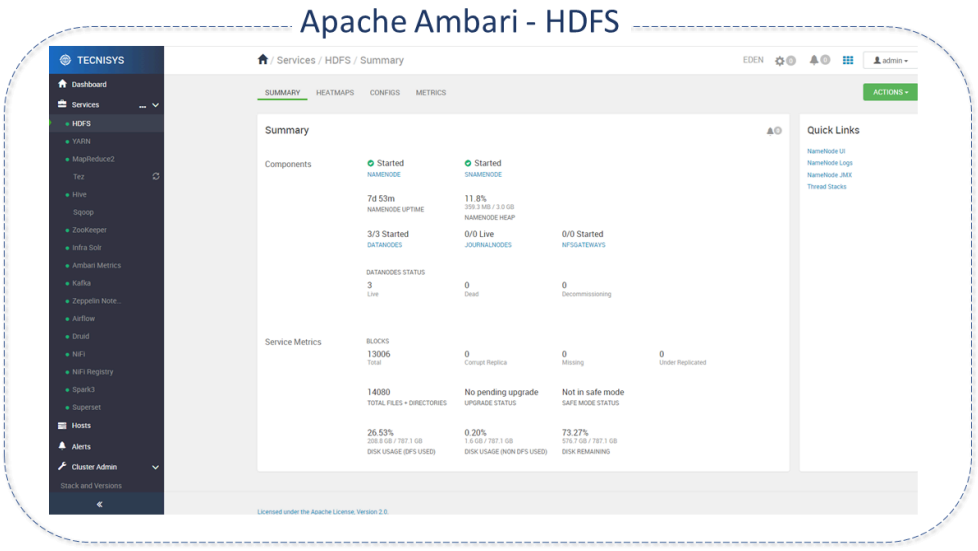


Figura 2 - Interface Ambari/HDFS

Arquitetura do Apache HDFS

© Copyright Tecnisys 2026. All rights reserved.



A arquitetura do HDFS é do tipo *mestre/escravo*, no qual um único mestre (o Namenode) controla a operação dos demais, os escravos (DataNodes).

HDFS divide um arquivo em blocos, que são armazenados como unidades independentes, o que facilita os processos de distribuição, replicação, recuperação e processamento.

- **Namenode:**
  - O Namenode é a peça central do sistema de arquivos HDFS. É ele quem realiza o mapeamento dos blocos para os DataNodes, determinando onde eles e suas réplicas devem ser armazenados e registrando a que arquivo pertence cada um. O NameNode atua através dos arquivos:
    - *fsimage:* que armazena quais blocos pertencem a cada arquivo
    - *edit log:* que armazena operações de escrita no sistema.

      > **NOTE**
      >
      > Quando o sistema é inicializado, o NameNode carrega toda informação do *fsimage* para sua memória, registrando o estado atual do sistema. Depois, registra as modificações por meio do arquivo *edit log* para atualizar o sistema.

  - Os aplicativos *cliente* conversam com o "Namenode" sempre que desejam localizar ou trabalhar com um arquivo(copiar/mover/excluir) e o *Namenode* responde às solicitações retornando uma lista de servidores onde os dados residem.
  - Quando o cliente executa uma operação de escrita, esta é registrada no arquivo de *edit log* e, depois, o *filesystem namespace* é modificado(onde ficam as informações sobre os arquivos). O *filesystem namespace* é armazenado localmente.
  - O NameNode executa operações de namespace do sistema de arquivos, como abrir, fechar, renomear arquivos e diretórios. Além disso, qualquer alteração no namespace é registrada pelo NameNode.

© Copyright Tecnisys 2026. All rights reserved.



- Um *Cluster* HDFS consiste em um único Namenode, o que simplifica muito a arquitetura do sistema.

- A Comunidade elaborou algumas orientações que merecem destaque em relação ao NameNode:

  - Recomendações de Uso para o Namenode
  - Namenode não inicia
  - "*ConnectionRefused*"

- Na nova arquitetura, o *Cluster HA* (alta disponibilidade) consiste de NameNodes em 03 distintos estados: *active*, *standby* e *observer*.

- **Datanodes:**
  Os DataNodes são responsáveis por atender às solicitações de leitura e gravação dos clientes do sistema de arquivo e armazenar os blocos.

  - Os Datanodes ficam espalhados pelo *Cluster*, geralmente um por nó:

    - São eles que realizam a criação, exclusão e replicação de blocos, sob instruções do Namenode.

    - Os blocos geralmente são lidos através do disco, porém blocos frequentemente acessados podem estar em *Cache* dentro do DataNode. Desse modo, gerenciadores de escalonamento podem rodar tarefas nos nós onde o bloco está em *cache*, melhorando a performance. +

    - Como são considerados pontos de falha, periodicamente enviam relatórios de funcionamento para o Namenode.

      > **NOTE**
      >
      > Um *hearbeat* indica que o DataNode está em funcionamento. Um *block report* indica quais os blocos estão armazenados no DataNode. Esses dados permitem que o NameNode saiba, de antemão, quais DataNodes estão disponíveis para armazenamento e para leitura, evitando, assim, o direcionamento do cliente para DataNodes que não estão em funcionamento.

© Copyright Tecnisys 2026. All rights reserved.

Os Namenodes e Datanodes são peças de *software* projetadas para executar em máquinas comuns. Estas máquinas geralmente executam um sistema operacional (S.O.) *GNU_Linux*.

- Uma implantação tipica possui uma maquina dedicada que executa apenas o software NameNode. Cada uma das outras máquinas no *Cluster* executam uma instância do software *DataNode*.

- A arquitetura não impede a execução de vários DataNodes na mesma máquina, mas em uma implantação real este caso é muito raro.

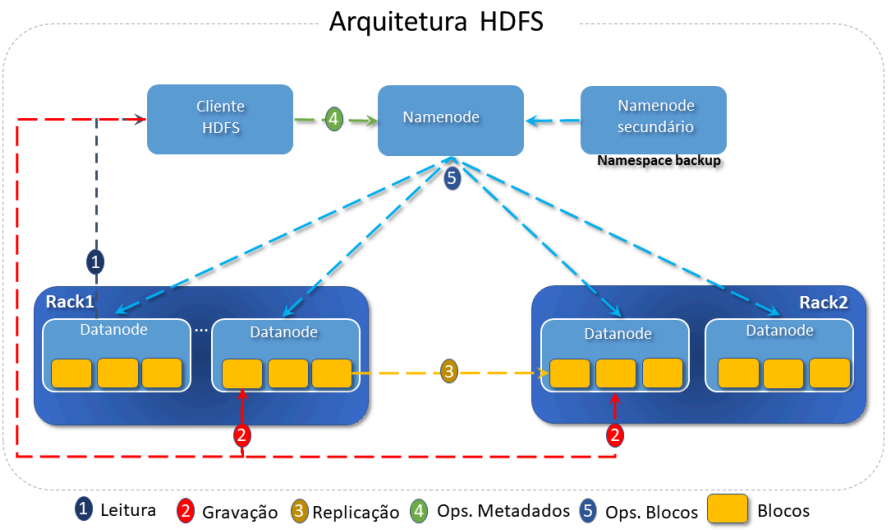


Figura 3 - Arquitetura HDFS

Recursos do Apache HDFS

- **NameSpace do Sistema de Arquivos:** O HDFS suporta a organização hierárquica tradicional de arquivos. Um usuário ou um aplicativo pode criar diretórios e armazenar arquivos dentro destes diretórios.

> **NOTE**
>
> A hierarquia de namespace do sistema de arquivos é semelhante à maioria dos outros sistemas de arquivos existentes: pode-se criar e remover arquivos, mover um arquivo entre diretórios ou renomeá-lo.

© Copyright Tecnisys 2026. All rights reserved.

- ***Userquotas*(Suporte a Quotas do Usuário)** Permite que o administrador defina cotas para o número de nomes e o volume de espaço usado para diretórios individuais. Quotas de nome e quotas de espaço operam de forma independente, mas a administração e implementação dos dois tipos é paralela.

- ***Access Permissions*(Suporte a Permissões de Acesso)** Implementa um modelo de permissões para arquivos e diretórios que compartilha muito do modelo POSIX.

- **Data Replication(Replicação de Dados)**:
  O HDFS foi projetado para armazenar grandes arquivos em máquinas de um grande *Cluster*. Cada arquivo é armazenado como uma sequência de blocos que são replicados para garantir a tolerância a falhas. O tamanho do bloco e o fator de replicação são configuráveis por arquivo.
  Todos os blocos de um arquivo, exceto o último, são do mesmo tamanho e os usuários podem iniciar um novo bloco sem preencher o último com o tamanho configurado, após o suporte para bloco de comprimento variado ter sido adicionado para *append* e *hsync*.
  Um aplicativo pode especificar o número de réplicas de um arquivo. Este número é chamado *repplication factor* e pode ser definido na criação e modificado posteriormente.
  O fator de replicação é armazenado pelo NameNode, que é, de fato, o responsável por tomar todas as decisões sobre replicação de dados.
  Por padrão, o número de replicas é 3 (três), e os blocos são separados de forma que o primeiro esteja em um *rack* distinto dos demais e o segundo no mesmo *rack* do terceiro, mas em nós separados.
  Desse modo o sistema consegue lidar com dois tipos de falhas: as falhas dos DataNodes e as falhas de *rack*.
  Além disso, essa política melhora as operações de escrita, pois reduz a *largura de banda*, considerando que entre *racks* distintos, ela é menor que dentro do mesmo *rack*, uma vez que os dados estão em dois *racks* distintos, não em três, que seria a política mais simples.
  O HDFS tenta satisfazer as requisições com uma réplica mais próxima do cliente.

© Copyright Tecnisys 2026. All rights reserved.

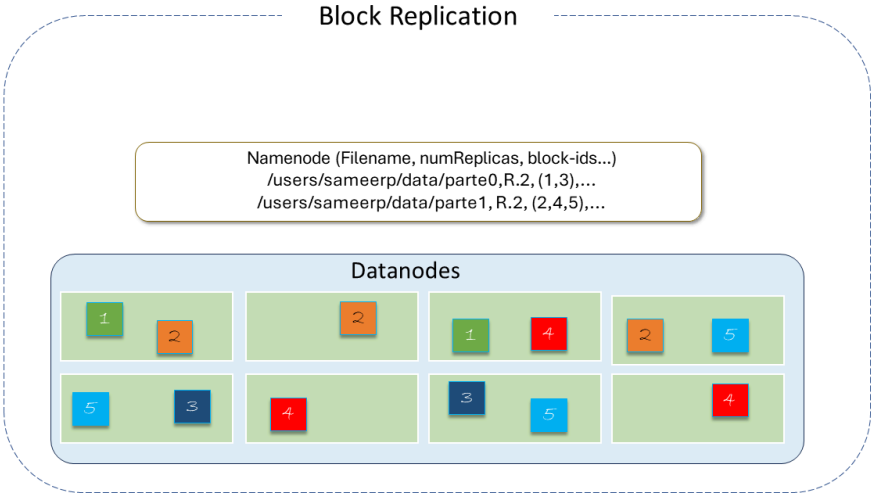


Figura 1 - Replicação de Blocos

- **Replica Placement(Posicionamento da réplica)**: A colocação das réplicas é critica para garantir confiabilidade e desempenho do HDFS. A Otimização de seu posicionamento distingue o HDFS da maioria dos outros sistemas de arquivos distribuídos. É um recurso que exige muito ajuste e experiência. A implementação da política de colocação de réplicas é um primeiro esforço nesta direção.
  Depois que o suporte para Storage Types and Storage Policies(tipos de armazenamento e políticas de armazenamento) foi adicionado ao HDFS, o NameNode leva em consideração a política para o posicionamento da réplica, além do conhecimento do rack.
  Detalhes sobre o assunto podem ser obtidos no item "Replicas Placement: The First Baby Steps", no site da comunidade.

- **Replica Selection:** Para minimizar o consumo global da largura de banda e latência, HDFS tenta satisfazer uma solicitação de leitura a partir de uma replica que esteja proxima do leitor. Se existir, no mesmo rack que o nó leitor, esta replica terá preferência para satisfazer a solicitação. Se o *Cluster* HDFS espalha-se por multiplos *Datacenters*, então a replica que está no *data center* local terá preferência sobre as remotas.

- **Políticas *Block Placement*:** Quando o fator de replicação é 3(três), a política de posicionamento do HDFS é colocar uma replica na máquina local, se o gravador estiver em um DataNode, caso contrário, em um datanode aleatório no mesmo rack que o gravador, outra replica em um nó em um rack remoto diferente, e a

© Copyright Tecnisys 2026. All rights reserved.

última em um nó diferente no mesmo rack remoto.
Quando o fator de replicação é maior que 3, a colocação das demais replicas são determinadas aleatoriamente mantendo o número de replicas por rack menor que o limite superior (calculado como: *replicas-1/racks+2*).
Adicionalmente, o HDFS oferece suporte a **quatro distintas políticas de colocação de blocos conectáveis**.
Os usuários podem escolher a política com base em sua infraestrutura e caso de uso. Por padrão, o HDFS oferece suporte a *BlockPlacementPolicyDefault*.

  - **Safemode**: Na inicialização, o NameNode entra em estado inicial denominado *Safemode*. A replicação de blocos não ocorre, nesta situação. O NameNode recebe mensagens *Heartbeat* e *Blockreport*(com a lista de blocos de dados que um DataNode está hospedado) dos DataNodes.
    Cada bloco tem um número especificado mínimo de replicas. Um bloco é considerado *replicado com segurança* quando o número mínimo de replicas do bloco de dados é verificado com o NameNode. Depois que um percentual configurável de blocos de dados replicados com segurança faz check-in com o NameNode (mais 30 segundos adicionais), o NameNode sai do estado Safemode e, em seguida, determina a lista de blocos de dados (se existir) que ainda possui menos do que o número especificado de réplicas. O NameNode então replica estes blocos para outros DataNodes.

- **HDFS Federation**: O HDFS Federation adiciona suporte a múltiplos NameNodes/NameSpaces ao HDFS.

- **Viewfs**: O ViewFs(View File System) provê uma forma de gerenciamento de múltiplos Namespaces (ou volumes de Namespaces). É particularmente útil para *Clusters* com multiplos NameNodes e ambém múltiplos NameSpaces, no *HDFS Federation*. É análogo ao *client side mount tables* em sistemas Unix-Linux. Pode ser usado para criar views personalizadsa de NameSpaces e também views comuns por *Cluster*.

- **Snapshots:** São cópias "pontuais" somente leitura do sistema de arquivos. Alguns casos de uso comuns de *snapshots* são o *backup* de dados, proteção contra erros do usuário e recuperação de desastres.

- ***Edits Viewer***: Permite a análise do arquivo de *log* de edições. Opera apenas em arquivos e não exige que o *Cluster* Hadoop esteja em execução.

© Copyright Tecnisys 2026. All rights reserved.



- **ImageViewer**: Ferramenta que "despeja" o conteúdo dos arquivos *fsimage* do HDFS em um formato "legível" e provê uma API *WebHDFS* somente leitura para analise e exame *offline* do NameSpace de um *Cluster* Hadoop.

- **Gerenciamento de Cache centralizado**: Permite a especificação de caminhos a serem armazenados em *cache* pelo HDFS.

- **Gateway NFS**: Permite que o HDFS seja "montado" como parte do sistema de arquivos local do cliente. Suporta NFSv3.

- **Criptografia transparente:** O HDFS implementa criptografia transparente de ponta a ponta. Depois de configurados, os dados lidos e gravados em diretórios HDFS especiais são criptografaos e descriptografados de forma "transparente", sem exigir alterações no código do aplicativo do usuário.

- **Multihomed Networks:** O HDFS suporta o *Multihomed Newworks*, onde os nós do *Cluster* são conectados com mais de uma interface de rede. Existem várias razões para usá-lo, como segurança, performance, e tolerância a falhas.

- **Suporte à armazenamento de memória:** O HDFS suporta a gravação na memória *off heap*(controlada pelo desenvolvedor) gerenciada pelos DataNodes.

- **Synthetic load generator:** Ferramenta de testes que permite verificar o comportamento do NameNode sob diferentes cargas de clientes.

- **Disk Balancer:** Ferramenta de *linha de comando* que distribui dados uniformemente em todos os discos do DataNode. É uma ferramenta diferente de Balancer,que cuida do balanceamento de dados no *Cluster*. O Disk Balancer opera criando um plano e executando este plano no DataNode. É ativado por padrão em um *Cluster*.

- **Guia de Administração do DataNode:** Ferramenta que permite operações como:

    - Estado de manutenção do DataNode, criado para permitir reparos/manutenções.
    - Desativação de DataNodes.
    - Recomissionamento.
- ***Router Federation*:** Adiciona uma camada de software para "federar" os Namespaces, permitindo que usuários acessem qualquer subcluster de forma

© Copyright Tecnisys 2026. All rights reserved.

transparente.

- ***Provided Storage*:** Permite que os dados armazenados fora do HDFS sejam mapeados e endereçados a partir do HDFS.

- **Observer NameNode:** Em um *Cluster* HDFS habilitado à Alta Disponibilidade (HA), existe um único NameNode Ativo (Active Namenode) e um ou mais NameNode(s) "standby".
  O NameNode ativo é responsável por servir todas as requisições dos clientes, enquanto o NameNode em "standby" apenas mantém as informações atualizadas sobre o NameSpace, assim como informações sobre a localização dos *blocos*, recebendo relatórios de todos os DataNodes.
  O recurso *Observer NameNode* endereça as funções acima por meio de um novo NameNode chamado "Observer NameNode". Da mesma forma que o NameNode "standby", o *Observer NameNode* se mantém atualizado em relação ao NameSpace e informções de localização do bloco.
  Mas além disso, tem a capacidade de produzir leituras consistentes, como o Active NameNode. Com as solicitações de leitura são uma grande parte em um ambiente típico, isto ajuda a balancear a carga do tráfego NameNode e melhorar a taxa de transferência geral (throughput).

Observações importantes

- Embora o HDFS siga a convenção de nomes do Filesystem , alguns caminhos e nomes (p.ex: /.reserved e .snapshot) são reservados. Recursos como criptografia transparente (transparent encryption) e snapshot usam caminhos reservados.
- Atualmente, o HDFS não suporta *hard links* (o link atuando como ponteiro para o *inode* de um arquivo ou diretório) ou *soft links*(ou link simbólico - atuando como ponteiro ou referencia para o nome do arquivo). No entanto sua arquitetura não impede a implementação destes recursos.

Detalhes do Projeto Apache HDFS

O HDFS foi construído na linguagem Java, altamente portável, o que o torna implementável em uma ampla variedade de máquinas. Qualquer máquina que suporta Java pode executar o software NameNode ou DataNode.

fonte(s): Hadoop Documentation

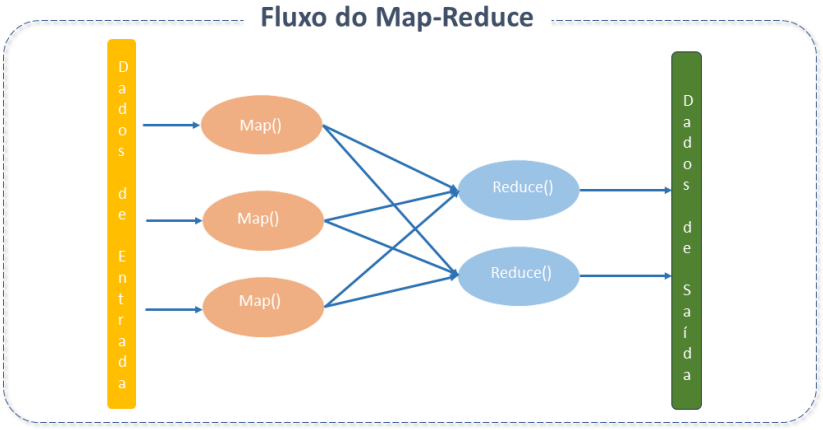
© Copyright Tecnisys 2026. All rights reserved.

MapReduce

Apache MapReduce é um *framework* desenvolvido para escrever aplicativos que processam grandes quantidades de dados em grandes *clusters* de hardware comum. Ele assegura confiabilidade, paralelismo, tolerância a falhas e facilita o *data locality* (possibilidade de um programa executar onde os dados estão armazenados).

O MapReduce foi desenvolvido a partir de uma tecnologia divulgada pelo Google, criada por Jeffrey Dean e Sanjay Ghemawat, para otimizar a indexação e catalogação de dados da web.

Apesar de o Hadoop ter evoluído desde sua versão inicial, o fluxo de alto nível do processador MapReduce manteve-se constante.



Filosofia do MapReduce

Arquitetura do Apache MapReduce

O MapReduce organiza o processamento em dois processos principais — Map e Reduce — com várias tarefas menores integradas ao fluxo do processo. Um trabalho MapReduce tipicamente divide o *dataset* de entrada em partes independentes

© Copyright Tecnisys 2026. All rights reserved.



processadas em paralelo pelas tarefas *Map*. Os resultados são então classificados e passados para as tarefas *Reduce*.

A entrada e a saída do *job* normalmente são armazenadas em um sistema de arquivos. Geralmente, os nós de computação e armazenamento são os mesmos, permitindo uma programação eficiente das tarefas nos nós onde os dados estão armazenados, resultando em maior largura de banda do *cluster*.

O *framework* MapReduce inclui um único *Resource Manager master*, um *NodeManager worker* por nó do *cluster* e um *MRAppMaster* por aplicação.

Os aplicativos especificam locais de entrada e saída e fornecem funções de *map* e *reduce* por meio de interfaces ou classes abstratas apropriadas. Esses e outros parâmetros do *job* formam a configuração do *job*.

O cliente submete o *job* (jar/executável, etc.) e a configuração para o *ResourceManager*, que gerencia a distribuição aos *workers*, o agendamento das tarefas e o monitoramento, fornecendo *status* e diagnósticos.

Entradas e Saídas

MapReduce funciona exclusivamente com pares chave-valor, tratando a entrada como um conjunto de pares e produzindo um conjunto de pares como saída. As classes de chave e valor devem ser serializáveis, implementando a interface *WritableComparable* para facilitar a ordenação.

Componentes do Processo

- **InputFileFormat**: O processo do MapReduce inicia com a leitura do arquivo armazenado no HDFS, que pode ser de qualquer tipo e cujo processamento é controlado pelo *Inputformat*.

- **RecordReader e *Input Split***: O arquivo é dividido em "partes" que são conhecidas como *input split*. Seus tamanhos são controlados pelos parâmetros *mapred.max.split.size* e *mapred.min.split.size*. Por default, o tamanho do *input split* é o mesmo que o do bloco e não pode ser mudado, exceto em casos muito específicos. Para arquivos com formato não-divisíveis, como .gzip, o *input split* será igual ao tamanho do arquivo.

© Copyright Tecnisys 2026. All rights reserved.



A função *RecordReader* é responsável por ler dados do *input split* armazenado no HDFS. O formato default é *TextInputFileFormat* e o delimitador do RecordReader é /n, o que significa que apenas uma linha será tratada como um registro pelo RecordReader. Algumas vezes o comportamento do RecordReader pode ser customizado, com a criação de um RecordReader proprio.

- **Mapper**: A classe *Mapper* é responsável pelo processamento do *input split*. A função *RecordReader* passa cada registro lido para a função *Map* do *Mapper*. O *Mapper* contém os métodos *setup* e *cleanup*.

    - O *setup* é executado antes do processamento do *Mapper* e, portanto, qualquer operação de inicialização deve ser feita dentro dele.
    - O método de limpeza *cleanup* é executado assim que todos os registros na *input split* forem processados e, portanto, qualquer operação de limpeza deve ser executada dentro dele.

  O *Mapper* processa os registros e emite a saída (output) usando o *object context*, que habilita o Mapper e o Reducer a interagir com outros sistemas Hadoop, permitindo, ainda, a comunicação entre *Mapper*, *Combiner* e *Reducer*.

    - Os pares de saída não precisam ser do mesmo tipo que os de entrada.
    - Os aplicativos podem usar o *counter* para relatar suas estatísticas.
    - O número de mapas geralmente é determinado pelo tamamho total de entradas (número total de blocos dos arquivos de saída).
- **Partitioner:** O *Partitioner* atribui um número de partição ao registro emitido pelo *Mapper* para que os registros com a mesma chave obtenham sempre o mesmo número de partição, assegurando que os registros com a mesma chave sempre irão para o mesmo *Reducer*.

- ***Shuffling* e *Sorting*:** O processo de transferir dados do *Mapper* para o *Reducer* é conhecido como *shuffling*. O *Reducer* inicia *threads* para leitura de dados da máquina *Mapper* e lê todas as partições que pertencem a ela para processamento usando o protocolo HTTP.

- **Reducer:** O número de *Reducers* que o Hadoop pode ter depende do número de saídas do *Map* e vários outros parâmetros, e isto pode ser controlado.
  O *Reducer* contém *reduce()* que é executado para cada chave única emitida pelo *Mapper*.

    - O número total de reduções geralmente é calculado como :

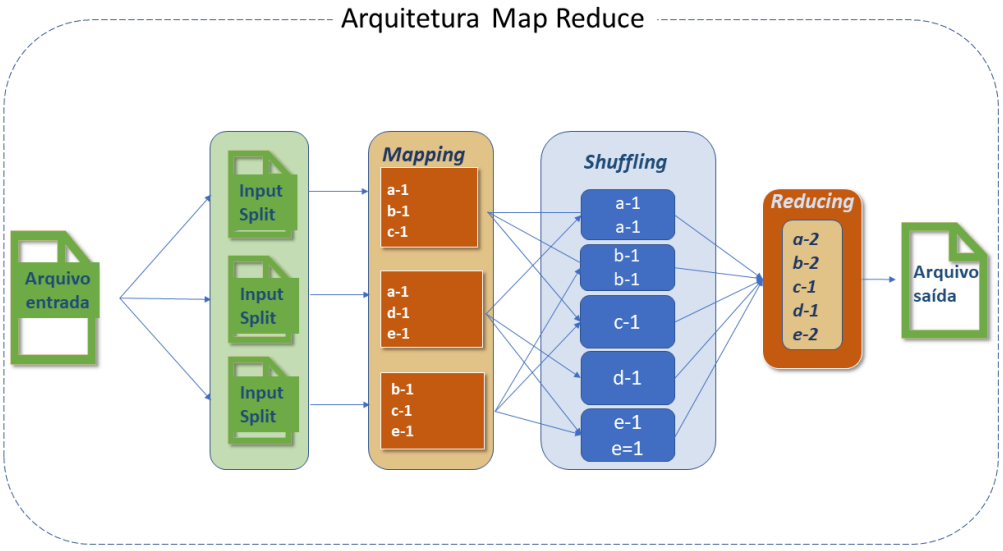
© Copyright Tecnisys 2026. All rights reserved.

```
(0,95 ou 1,75) * (num. de nós * núm máximo de contentores por nó.
```

> **NOTE**
>
> Com 0,95 todas as reduções podem ser iniciadas imediatamente e começar a transferir as saídas do mapa à medida que terminam. Com 1,75 os nós mais rápidos terminarão sua "primeira rodada" de reduções e lançarão uma segunda onda, fazendo um melhor trabalho de balanceamento de cargas.

- **Combiner**: Também chamado *mini reducer* ou *localized reducer* é um processo executado nas máquinas *Mappers* que obtém a chave intermediária emitida pelo *Mapper* e aplica a função *reduce* definida pelo usuário do *Combiner* na mesma máquina.
  Para cada *Mapper* existe um *Combiner* disponível na máquina *Mappers*. O *Combiner* reduz significativamente o volume de *data shuffling* entre Mapper e *Reducer* e ajuda, portanto, na melhoria da performance. Entretanto, não há garantia que sejam executados.

- **Output format:** Traduz o par final *chave/valor* da função *Reduce* e o grava em um arquivo, no HDFS, por meio do *record writer* . Por default, os valores chave de saída são separados por *tab* e os registros são separados por um caracter *newline*. Isto pode ser modificado pelo usuário.



Arquitetura do Apache MapReduce

© Copyright Tecnisys 2026. All rights reserved.



Padrões do Apache MapReduce

Existem soluções de template desenvolvidas por pessoas que resolveram problemas específicos, e que podem ser reutilizados:

- Padrões de "sumarização":
  - Contagem de palavras.
  - Minimo e máximo
- Padrões de "filtragem":
  - Algoritmo reduce "top-k": Algoritmo popular do MapReduce em que os mapeadores são responsaveis por emitir os registros tok-k em seu nível e, em seguida, o redutor filtra os registros top-k de todos os recebidos do mapeador.
- Padrões de "junção":
  - Reduce side join: processo onde a operação de "junção" é feita na fase "reducer". Mapper lê o dado de entrada que é combinado baseado em uma coluna comum ou uma chave de junção.
  - Junção Composta:
    - Classificação e particionamento

Boas Práticas para o Apache MapReduce

- **Configuração do Hardware:**
  - A configuração do Hardware é muito importante para a performance. Um sistema com mais memória sempre apresentará melhor desempenho.
  - A largura de banda também é crítica, uma vez que os job MapReduce podem exigir *shuffling* de dados de uma máquina para outra.
- ***Tunning* do Sistema Operacional:**
  - *Transparent Huge Pages (THP):* Máquinas usadas no Hadoop devem ter o THP desabilitado. O THP não funciona bem em um *cluster* Hadoop e causa alto custo de CPU. A recomendação é desabilitá-lo em cada nó do job.
  - *Evitar swapping de memória desnecessária:* No Hadoop, o *swapping* pode afetar a performance de *jobs* e deve ser evitado a menos que seja

© Copyright Tecnisys 2026. All rights reserved.

absolutamente necessário. A configuração de *swappiness* pode ser definida como 0(zero).

- *Configuração da CPU:* Na maioria dos Sistemas Operacionais, a CPU é configurada para economizar energia e não é otimizada para sistemas como o Hadoop. Por padrão, o *scaling governor* está configurado para o modo de economia de energia e precisa ser alterado com o seguinte comando:

Terminal input

```
cpufreq-set -r -g
```

- **Ajustes na rede:** O *shuffling* consome um tempo significativo do Hadoop pois demanda a conexão *master* e *worker* com muita frequência. O *net.core.somax.conn* deve ser definido com um valor mais alto, o que pode ser feito adicionando ou editando `/etc/sysctl.conf` no arquivo `net.core.somaxconn=1024`.

- **Escolha do sistema de arquivos:**

  - A distribuição Linux vem com um sistema de arquivos padrão projetado para cargas intensas de E/S, o que pode impactar de forma significativa o desempenho do Hadoop. A distribuição mais recente do Linux vem com o `EXT4` como sistema de arquivos padrão, que funciona melhor que o `EXT3`.
  - O Sistema de arquivos registra o último horário de acesso para cada operação de leitura no arquivo, e, portanto, causa uma gravação em disco para cada leitura. Esta configuração pode ser desabilitada adicionando um atributo *noatime* à opção *file system mount*. Alguns casos de uso observaram uma melhoria de desempenho em mais de 20% com o *noatime*.

- **Otimizações:**

  - *Combiner:* O *shuffling* dos dados pela rede pode ser caro pois a transferência de mais dados sempre consumirá mais tempo de processamento.
    O *reduce* não pode ser usado em todos os casos de uso, mas na maioria dos casos podemos usar o *Combiner*, que reduz o tamanho dos dados que serão transferidos pela rede durante o *shuffling* pois atua como um *mini reduce* e é executado na máquina dos *Mappers*.

© Copyright Tecnisys 2026. All rights reserved.



- *Compactação do Map output:* O *Mapper* processa a saída e a armazena no disco local.
  Quando gera uma grande quantidade de saídas, este resultado intermediário pode ser compactado com a função *LZO*, reduzindo I/O no disco durante o *shuffling*.
  Isto é feito definindo o `mapred.compress.mapoutput` como `true`.

- *Filtro de registros:* Filtrar registros no lado do *Mapper* resulta em menos dados gravados no disco local e maior rapidez nas próximas etapas (com menos dados para operar).

- *Evitando muitos arquivos pequenos:* Arquivos muito pequenos podem exigir muito tempo para executar.
  O HDFS armazena estes arquivos como um bloco separado, o que pode sobrecarregar o processamento dos arquivos com a inicialização de muitos *Mappers*.
  É uma boa prática compactar arquivos pequenos em um único arquivo grande e então executar o MapReduce nele.
  Em alguns casos isto pode trazer uma otimização em 100% da performance.

- *Evitando formato de arquivo não divisível:* O formato não divisível (p.ex: gzip) é processado de uma só vez.
  Se forem arquivos muito pequenos, consumirá muito tempo, pois para cada arquivo, um Mapper será iniciado.
  A melhor prática é usar um formato divisível como Texto, AVRO, ORC, etc.

- **Configurações *Runtime***

  - *Memória Java: Map* e *Reduce* são processos JVM que, portanto, usam memória JVM para execução.
    O tamanho da memória pode ser ajustado na propriedade *mapred.child.java.opts*.

  - *Memória Map spill:* Os registros de saída do *Mapper* são armazenados num *buffer* circular cujo tamanho default é 100MB.
    Quando a saída excede 70% deste tamanho, os dados serão levados para o disco.
    Para aumentar a memória do *buffer*, use a propriedade *io.sort.mb*.

© Copyright Tecnisys 2026. All rights reserved.



- *Ajuste no Map:* O número de *Mappers* é controlado pelo *mapred.min.split.size*. Se existirem muitas tarefas executando uma após a outra, é ideal configurar *mapred.job.reuse.jvm.num.tasks* para -1.

> **WARNING**
>
> Isto deve ser usado com muito cuidado pois em caso de tarefas com longa execução, a sobrecarga de JVM não aumentará o desempenho, pelo contrário.

- **Otimização do File System:**
  - *Mount option:* Existe algumas opções de montagem eficientes para *clusters* Hadoop, como por exemplo o *noatime* configurado para Ext4 e XFS.
  - *Tamanho do bloco HDFS:* O tamanho do bloco é importante no desempenho do NameNode e da execução do job.
    O Namenode mantém os metadados para cada bloco que armazena no Datanode e, portanto, ocupa muita memória em tamanhos de bloco muito menores que o recomendado.
    O valor recomendado para *dfs.blocksize* deve estar entre 134.217.728 e 1.073.741.824.
  - *Leitura short circuit:* A operaçao de leitura do HDFS passa pelo DataNode, que, após a solicitação do cliente, envia os dados do arquivo pelo *socket* TCP para o cliente.
    Na leitura *short circuit* o cliente lê diretamente o arquivo, e, portanto, ignora o DataNode.
    Isto só acontece se o cliente estiver no mesmo local que os dados.
  - *Stale Datanode:* O Datanode envia um *heartbeat* para o Namenode em intervalos regulares para informar que está ativo.
    Para evitar enviar solicitações de leitura e gravação para os Datanodes inativos, a adição das propriedades abaixo no *hdfs-site.xml* é eficiente:

Terminal input

```
_dfs.namenode.avoid.read.stale.datanode=true
```

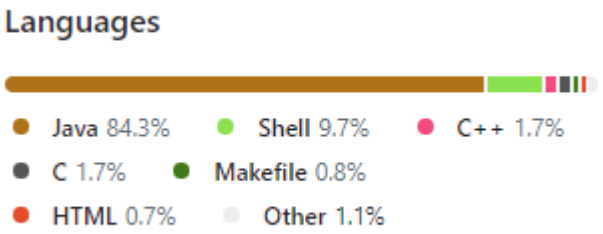
© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
_dfs.namenode.avoid.write.stale.datanode=true
```

Detalhes do Projeto Apache MapReduce

Embora o Hadoop seja implementado em Java, os aplicativos MapReduce podem usar qualquer linguagem que suporte Hadoop *Streaming* ou *Hadoop Pipes*.



Linguagens do MapReduce

Fontes:

- Hadoop.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Hive

Analytics



Um banco de dados é um conjunto de dados pertencentes ao mesmo contexto, armazenados sistematicamente dentro de uma estrutura construída para suportá-los. Nesta estrutura convivem as regras de negócio necessárias para atingir objetivos específicos.

Um banco de dados analítico é um tipo de banco de dados criado para armazenar, gerenciar e consumir dados, projetado para soluções específicas de Análise de Negócio, Big Data e BI.

Ao contrário de um banco de dados típico, que armazena dados por transações ou processo, um banco de dados analítico armazena dados históricos, em métricas de negócio. É baseado em modelos multidimensionais que usam as relações com os dados para gerar matrizes multidimensionais chamadas "cubo". Estes modelos podem ser consultados diretamente combinando suas dimensões, evitando consultas complexas que seriam realizadas em bancos de dados convencionais.

© Copyright Tecnisys 2026. All rights reserved.

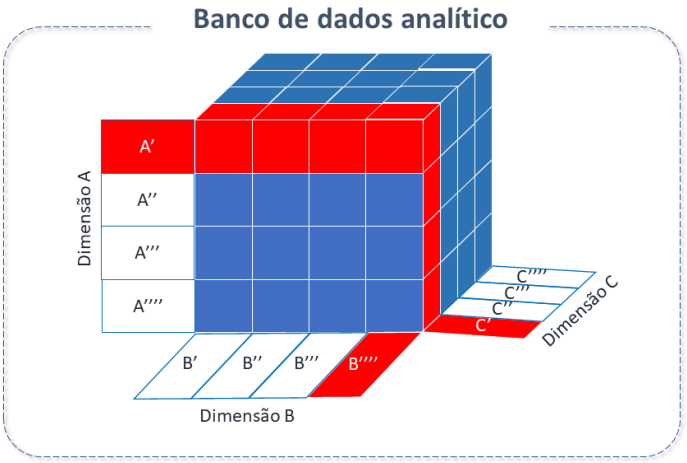


Figura 1 Banco de Dados Analítico

Características do Apache Hive

O *Apache Hive* é um banco de dados analítico (*Datawarehouse*) baseado no APACHE Hadoop, criado para facilitar o trabalho dos usuários data-warehouse com grande conhecimento em *queries* SQL, mas que encontram dificuldade para adotar Java ou outras linguagens.

Fornece interface semelhante a *SQL* entre o usuário e o sistema de arquivos distribuídos do Hadoop-HDFS.

Foi projetado para facilitar a sumarização, análise, consulta, leitura, gravação e tratamento de grandes conjuntos de dados.

Com ele, tornou-se possível definir tabelas com os dados armazenados no HDFS e então executar *queries* para transformação ou geração de relatórios.

- O *Apache Hive* oferece funcionalidades padrão *SQL*, incluindo muitos recursos SQL 2003, SQL 2011, SQL 2016 e posteriores, para análise.
- O *Apache Hive* também pode ser estendido por meio de "funções definidas pelo usuário (UDFs), agregações definidas pelo usuário (HDAFs) e funções de tabela definidas pelo usuário (UDTFs)".
- Não existe um "formato Hive" único. O Hive traz conectores integrados para:

© Copyright Tecnisys 2026. All rights reserved.

    - Arquivos de texto com valores separados por vírgulas e tabulações (CSV/TSV)
    - "Apache Parquet" (formato de armazenamento colunar disponível para qualquer projeto no ecossistema Hadoop)
    - *Apache ORC* (formato de arquivo colunar autodescritivo com reconhecimento de tipo projetado para cargas de trabalho do Hadoop)
    - outros formatos.
- O usuário pode estender o Hive com conectores para outros formatos a partir dos seguintes perfis:

	TextFile	SequenceFile	RCFile
Tipo de Dado	Apenas Texto	Texto/binário	Texto/binário
Armazenamento interno	Baseado em Linhas	Baseado em Linhas	Baseado em Colunas
Compressão	Baseado em Arquivo	Baseado em Bloco	Baseado em Bloco
Divisível	SIM	SIM	SIM
Divisível após compressão	NÃO	SIM	SIM

Figura 2 - Formatos arquivos HIVE

- As operações de consulta e armazenamento são semelhantes aos SGBD tradicionais. Entretanto, existem muitas diferenças na estrutura e funcionamento do Hive.

- *Apache Hive* não foi projetado para processamento de transações online (OLTP). É melhor usado para tarefas tradicionais de *Data Warehousing*.

- Foi projetado para maximizar a escalabilidade, desempenho, extensibilidade, tolerância a falhas e acoplamento flexível, com seus formatos de entrada.

- A simplicidade do *Hive query language (HQL)* tem ajudado o Hive a ganhar popularidade na comunidade Hadoop, sendo usado em muitos projetos por todo o mundo. HQL provê comandos de *explain* e *analyze* que podem ser usados para checar e identificar a performance de queries. Adicionalmente, os logs do Hive contêm informação detalhada para investigação de performance e solução de problemas.

© Copyright Tecnisys 2026. All rights reserved.

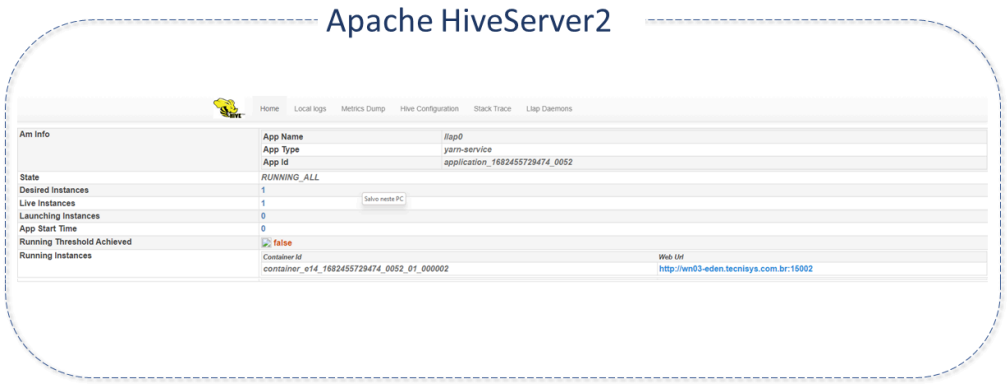


Figura 3 - Interface HiveServer2

Hive Metastore

O **Hive Metastore** é um componente da arquitetura do Apache Hive responsável pelo catálogo de metadados. Ele não executa consultas; essa função fica com os motores de processamento. Seu papel é guardar as definições lógicas que descrevem os dados: bancos de dados, tabelas, colunas, tipos, partições e localização física.

Os dados das tabelas permanecem no armazenamento configurado, como HDFS, sistemas de arquivos ou armazenamento de objetos. O Metastore mantém o mapa entre o nome lógico da tabela e os arquivos onde os dados estão armazenados, permitindo que clientes interpretem esses arquivos como tabelas.

Por isso, o Hive Metastore também é usado fora do Hive em plataformas de dados com Spark, Trino, Iceberg ou Delta Lake. Nesses cenários, ele funciona como catálogo compartilhado: diferentes motores consultam o mesmo serviço para descobrir tabelas, esquemas, partições e localizações.

Arquitetura do Apache Hive

Hive consiste em múltiplos componentes, sendo os principais:

- **Cliente Hive:** O Hive provê diferentes drives para comunicação com distintos tipos de aplicações.
    - Para aplicações *Thrift based*, disponibiliza o *Thrift client*.
    - Para aplicativos Java, provê *Drivers JDBC*.
    - Para outros tipos de aplicações, provê drives *ODBC*.

© Copyright Tecnisys 2026. All rights reserved.



- **Metastore server**: Armazena metadados para cada tabela, como esquema e localização, incluindo os metadados da partição, o que ajuda o *driver* a rastrear o progresso de vários conjuntos de dados distribuídos pelo *Cluster*.

    - Mantém detalhes sobre tabelas, partições, esquemas, colunas, etc. Os dados são armazenados em formato RDBMS tradicional.
    - Os metadados ajudam o *driver* a acompanhar os dados.
    - Um servidor de *Backup* replica regularmente os dados que podem ser recuperados em caso de perda de dados.
    - Provê interface *Thrift Service* para acesso às informações e metadados.
- ***Driver***: É responsável por receber as consultas Hive submetidas pelos clientes.

    - Inicia a execução da instrução criando sessões e monitora o ciclo de vida e o progresso da execução.

    - Armazena os metadados necessários gerados durante a execução de uma instrução *HIVEql*.

    - O *driver* também atua como ponto de coleta de dados ou resultados de consulta obtidos após a operação *reduce*.
      Possui quatro componentes:

        - ***Parser***: Responsável por "checar" erros de sintaxe em queries. É o primeiro passo da execução da query e, caso encontre irregularidades, retorna um erro para o cliente via *Driver*.
        - ***Planner***: Queries *parsed* com sucesso são encaminhadas ao *planner* que gera os planos de execução, usando tabelas e outras informações de metadados do metastore.
        - **Otimizador**: Responsável pela análise do plano e geração de um novo plano DAG otimizado. A otimização pode ser feita em *joins*, *reducing suffling data*, etc., visando otimização da performance.
        - ***Executor***: Uma vez *parser*, *planner* e *otimizador* tenham finalizado suas tarefas, o *executor* iniciará a execução do *job* na ordem das dependências. O Plano otimizado é comunicado a cada tarefa usando um arquivo. O *executor* cuida do ciclo de vida das tarefas e monitora sua execução.

© Copyright Tecnisys 2026. All rights reserved.

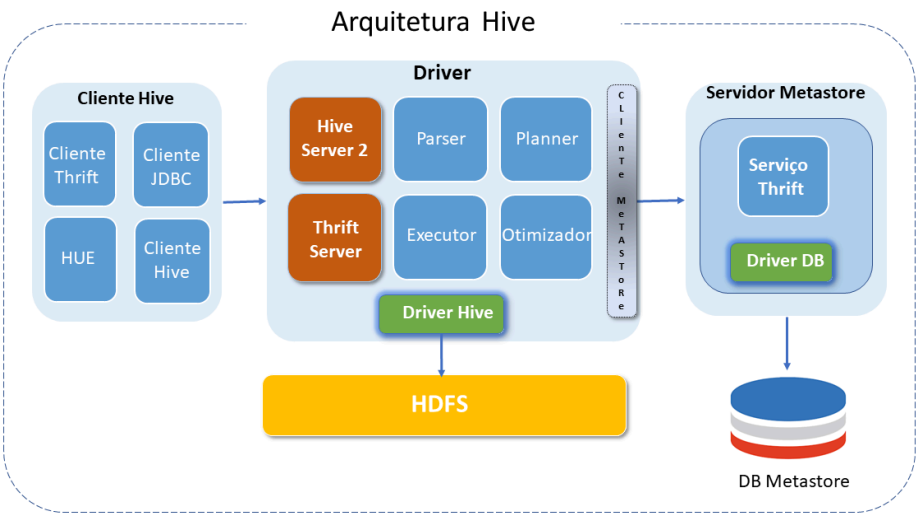


Figura 4 - Arquitetura Hive

Interação Hive com Hadoop

Em linhas gerais, a interação entre Hive e Apache Hadoop se dá da seguinte forma:

"1 " - O UI chama a *interface de execução* do *Driver*.
"2 " - O *Driver* cria um identificador de sessão para a consulta e a envia ao compilador para que seja gerado um plano de execução.
"3 e 4 " - O *compilador* obtém os metadados necessários no *metastore*, que serão usados para verificação do tipo de expressões na árvore de consulta e para remover partições com base nos predicados da consulta (quando um resultado booleano é esperado).

- O Plano gerado pelo *compilador* é um DAG de estágios sendo cada estágio um *job map* ou *reduce*, uma operação de metadados ou no HDFS.
- Para os estágios *Map/reduce*, o plano contém árvores de operador *map* (árvores de operador que são executados nos *mappers*) e uma árvore de operador *reduce* (para operações que usam *reducers*).

"6 " - O *Engine de execução* submete estes estágios aos componentes apropriados (6, 6.1, 6.2, 6.3).

- Em cada *map* ou *reduce*, o *desserializador* associado com a tabela ou saídas intermediárias é usado (para ler as linhas dos arquivos HDFS que são passadas pela árvore do operador associado).

© Copyright Tecnisys 2026. All rights reserved.

A *saída* é gravada em um arquivo HDFS temporário por meio do *serialializador* (o que acontece no *Mapper* caso a operação não necessite de *reduce*).

Os arquivos temporários são usados para fornecer dados para mapear/reduzir etapas subsequentes do plano.

Para operações DML, o arquivo temporário final é movido para o local da tabela.

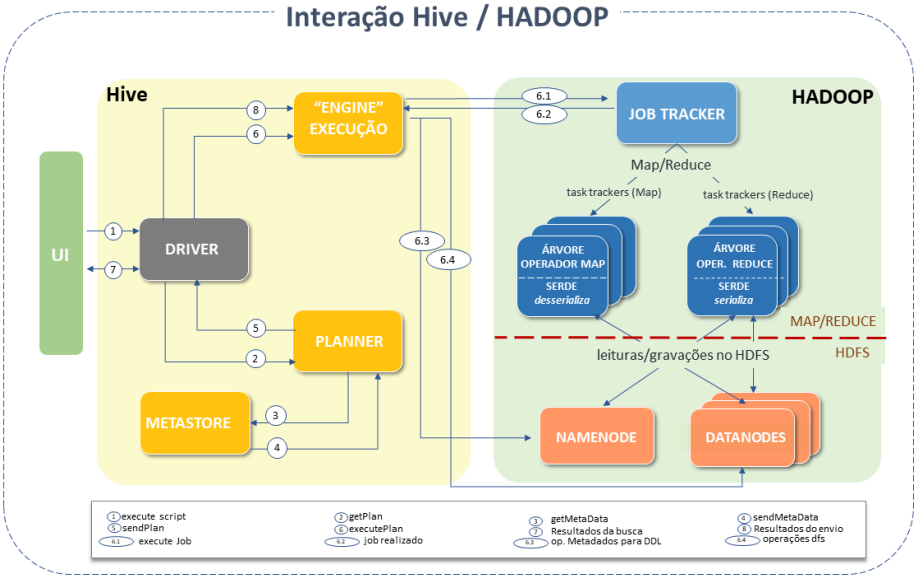


Figura 5 - Interação Hive-Hadoop

Como o Apache Hive Funciona

- **HCatalog**: É uma camada de gerenciamento de tabela e armazenamento para Hadoop que permite ferramentas como o Hive e MapReduce lerem dados de tabelas. Construído sobre o serviço Hive Metastore, ele suporta formatos de arquivos para os quais o Hive SerDe pode ser realizado.

  O HCatalog habilita a visualização de dados como tabelas relacionais, sem se preocupar com a localização ou o formato dos dados.
  Suporta formatos como text file, ORC file, sequence file, e RCFile, e SerDe pode ser escrito para formatos como AVRO (um formato de dados de código aberto que possibilita o agrupamento de dados serializados com o esquema de dados no mesmo arquivo).
  Tabelas HCatalog são "imutáveis", o que significa que os dados na tabela e na partição não podem ser anexados. No caso de tabelas particionadas, os dados podem apenas ser anexados a uma nova partição, sem afetar a partição antiga.

© Copyright Tecnisys 2026. All rights reserved.



As interfaces do HCatalog para aplicações MapReduce são *HCatInputFormat* e *HCatOutputFormat*, ambas compatíveis com *InputFormat* e *OutputFormat* do Hadoop. HCatalog está evoluindo novas interfaces para interagir com outros componentes do ecossistema Hadoop.
- **WebHCat** fornece um serviço para execução de tarefas Hadoop MapReduce (ou YARN), Pig e Hive. Também permite a execução de operações de metadados do Hive usando uma interface HTTP (REST-style).

Organização dos Dados

Por padrão, o Hive armazena metadados em um banco de dados Apache Derby embutido. Outros bancos de dados cliente-servidor podem ser usados opcionalmente, como o Postgree.

Os primeiros quatro formatos de arquivo suportados pelo Hive eram apenas texto simples, arquivos sequenciais, ORC, e RCFile. Apache Parquet pode ser lido via plug-in em versões posteriores a 0.1, originalmente iniciando em 0.13. A representação compactada de dados do Parquet (colunar) reduz a quantidade de dados que o Hive tem que percorrer e, consequentemente, o tempo de execução das consultas.

Plugins adicionais suportam a consulta do Blockchain Bitcoin.

Unidades de Dados

Hive é organizado em:

- **Banco de Dados**: Os Namespaces existem para evitar conflitos de nomes em tabelas, views, partições, colunas e tudo o mais. Databases também podem ser usados para reforçar a segurança para um usuário ou grupo de Usuários.

- **Tabelas**: Unidades homogêneas de dados que possuem o mesmo esquema.

- **Partições**: Cada tabela pode ter uma ou mais chaves de partição que determinam como os dados são armazenados. Partitions além de ser unidades de armazenamento, também habilitam o usuário a identificar, de forma eficiente, as linhas que satisfazem um critério específico. Cada valor único de uma chave de partição define uma partição da tabela.

- ***Buckets* ou *Clusters***: Dado em cada partição pode ser dividido em "porções" baseado no valor da função hash de alguma coluna da tabela. Por exemplo, uma tabela pode ser "quebrada" pelo userid, a qual é uma das colunas da tabela.

© Copyright Tecnisys 2026. All rights reserved.



Recursos do Apache Hive

Construído sobre o Apache Hadoop, o Hive fornece os seguintes recursos:

- **Acesso via SQL**, facilitando tarefas de armazenamento de dados, como extração/transformação/carga (ETL), geração de relatórios e análise de dados.
- Suporte à simultaneidade e autenticação de vários clientes com **Hive-Server 2 (HS2)**, projetado para fornecer melhor suporte a cliente de API aberta, como o *JDBC* e *ODBC*.
- **Suporte completo à ACID** para tabelas *ORC* e de inserção a todos os outros formatos.
- **Suporte à inicialização e à replicação incremental** para *Backup* e recuperação de dados.
- **Repositório central de metadados** para tabelas e partições Hive** em um SGBD relacional com acesso a estas informações por meio de API do serviço MetaStore.
- **Mecanismo para estabelecer estrutura em uma variedade de formato de dados**.
- **Otimizador de consultas baseado em custos (CBO)**.
- **Acesso a arquivos armazenados diretamente no Apache HDFS ou outros, como o Apache Hbase**.
- **Execução de consulta via Apache Tez, Apache Spark, ou Mapreduce**.
- **Linguagem procedural com HPL-SQL**.
- **Recuperação de consultas em subsegundos via Hive LLAP e Apache YARN**.
- **Beeline-Command Line Shell** HiveServer2 suporta Beeline Shell, um cliente *JDBC* baseado em SQL CLI que opera tanto em modo embutido (com um Hive integrado - semelhante ao Hive CLI) quanto remoto (conectando-se a um processo HS2 separado via Thrift).

Principais Diferenças entre Hive e os SGBD Tradicionais

© Copyright Tecnisys 2026. All rights reserved.

Nos SGBD tradicionais, um esquema é aplicado a uma tabela quando os dados são carregados nela.
Isso permite que o SGBD verifique se os dados inseridos seguem a representação da tabela, conforme especificado em sua definição. Este "design" é chamado "esquema na gravação".
O Hive não verifica os dados em relação ao esquema da tabela na gravação. Ele faz verificações de tempo quando os dados são lidos. Este modelo é denominado "esquema em leitura". As duas abordagens têm suas vantagens e desvantagens.
A verificação durante a carga adiciona sobrecarga, onerando o tempo de carga de dados, mas garantindo que os dados não estejam corrompidos.
A detecção antecipada garante o tratamento antecipado de exceções. Como as tabelas são forçadas a corresponder ao esquema durante/após a carga dos dados, tem melhor desempenho em tempo de consulta.
O Hive, por outro lado, pode carregar dados dinamicamente sem verificação de esquema, garantindo uma carga inicial rápida, mas com desempenho mais lento no momento da consulta.
As transações são operações fundamentais em SGBD tradicionais.
Como qualquer RDBMS típico, o Hive suporta as quatro propriedades das transações (ACID): Atomicidade, Consistência, Isolamento, Durabilidade. As transações foram incluídas no Hive 0.13, mas limitadas apenas ao nível de partição.
A versão 0.14 do Hive foi totalmente adicionada para suportar propriedades ACID completas. A partir desta versão, é possível realizar diferentes transações em nível de linha, como insert, delete, update.
A ativação destes comandos requer a fixação de valores apropriados para as propriedades de configuração, como hive.support.concurrency, hive.enforce.bucketing, hive.exec.dynamic.partition.mode

Segurança do Apache Hive

A versão 0.7.0 do Hive adicionou integração com a segurança do Hadoop, que, por sua vez, passou a usar o suporte de autorização do Kerberos para fornecer segurança.
O Kerberos permite a autenticação mútua entre cliente e servidor. Nesse sistema, a solicitação do cliente para um ticket é passada junto com a solicitação.
Permissões para arquivos recém-criados no Hive são ditadas pelo HDFS. O modelo de autorização do sistema de arquivos distribuídos do Hadoop usa três entidades: usuário, grupo e outras com três permissões: ler, gravar, executar. As permissões padrão para arquivos recém-criados podem ser configuradas alterando o valor unmask para a variável de configuração Hive hive.files.umask.value.

© Copyright Tecnisys 2026. All rights reserved.

Boas Práticas para Apache Hive

- **Particionamento de Tabelas:** O Apache Hive resolve a ineficiência da execução de *jobs* MapReduce com grandes tabelas oferecendo um esquema de partições automático no momento da criação da tabela.
  Neste método, todos os dados da tabela são divididos em múltiplas partições, cada qual correspondendo a(os) valor(es) específico(s) da(s) coluna(s) da partição, que é(são) mantido(s) em registro(s) da tabela HDFS (como sub-registro(s)).
  Uma consulta com filtragem de partição carregará apenas dados das partições específicas (subdiretórios), trazendo mais velocidade ao processo. A seleção da chave de partição é um fator importante, deve sempre ser um atributo de baixo "cardinal" para evitar sobrecargas.

  > **NOTE**
  >
  > Alguns atributos comumente usados como chaves de partição são:
  >
  > - Partições por data e hora: data-hora, ano-mês-dia (mesmo horas)
  > - Partições por local: país, território, cidade, estado.
  > - Partições por lógica de negócio: departamento, clientes, região de vendas, aplicativos.

- **Desnormalização**: A normalização é um processo padrão usado para modelar tabelas de dados lidando com redundâncias e outras anomalias.
  Junções são operações caras e complicadas e geram problemas de desempenho.
  É uma boa ideia evitar estruturas de tabelas altamente normalizadas, pois exigem a junção de consultas.

- **Compactação da saída Map/Reduce**: A compactação reduz significativamente o volume de dados intermediários e minimiza a quantidade de transferência de dados entre Mapper e Reducer.

  - A compactação pode ser aplicada à saída do Mapper e do Reducer individualmente. Importante lembrar que arquivos compactados com gzip não podem ser divididos. Significa que deve ser aplicado com cautela.
  - O tamanho do arquivo não deve ser maior do que algumas centenas de Mb, sob risco de produzir um trabalho desequilibrado.
  - Opções de codec de compressão podem ser snappy, izo, bzip, etc.

© Copyright Tecnisys 2026. All rights reserved.



- ***Map Joins* (junções de mapa)**: *Map Joins* são eficientes se a tabela "no outro lado do join" for pequena o suficiente para residir em memória.
  Com o parâmetro `hive.auto.convert.join=true`, o Hive tenta o map join automaticamente.
  Ao usar este parâmetro, é adequado certificar-se de que o auto-convert esteja habilitado no ambiente Hive.
  É essencial, ainda, o parâmetro `hive.enforce.bucketing=true`. Esta configuração fará com que o Hive reduza os ciclos de scan para encontrar uma chave específica porque o bucketing garantirá que a chave esteja em um bucket específico.

- **Seleção do Formato de Entrada**: Formatos de entrada têm um papel crítico no desempenho do Hive. O tipo texto, por exemplo, não é adequado para um sistema com alto volume de dados, pois estes tipos legíveis de formatos ocupam muito espaço e trazem sobrecarga à análise.

  - Para resolver isto, o Hive traz formatos de entrada colunares como RCFile, ORC, etc. Estes formatos permitem reduzir operações de leitura em consultas analíticas, viabilizando o acesso a cada coluna individualmente.
  - Outros formatos binários como Avro, arquivos de sequência, Thrift e ProtoBuf podem ajudar.
- **Execução Paralela**: Hadoop pode executar *jobs* MapReduce em paralelo e várias consultas executadas no Hive usam este paralelismo automaticamente.

  - Entretanto, consultas únicas e complexas do Hive geralmente são convertidas em vários *jobs* MapReduce, que, por sua vez, são executados sequencialmente por padrão.
  - Alguns dos estágios MapReduce de uma consulta não são interdependentes e podem ser executados em paralelo, aproveitando a capacidade "disponível" do cluster, melhorando sua utilização e reduzindo o tempo geral de execução.
  - A configuração Hive para mudar este comportamento é alternar o sinalizador `set hive.exec.parallel=true`.
- **Vetorização**: A execução de consultas vetorizadas melhora o desempenho do Hive.

  - É uma técnica que trabalha com o processamento de dados em lotes, em vez de uma linha por vez, o que resulta em uma eficiente utilização da CPU.
  - Para ativar a vetorização, o parâmetro adequado é `set hive.vectorized.execution.enabled=true`.

© Copyright Tecnisys 2026. All rights reserved.

- **Teste de Unidade**: O teste de unidade determina se a menor parte testável do código funciona. Detectar problemas antecipadamente é muito importante.

    - O Hive permite testar unidades UDFs, SerDes, Scripts de streaming, Consultas do Hive, etc., por meio de testes rápidos em modo local.
    - Existem várias ferramentas disponíveis que auxiliam o teste de consultas Hive, como HiveRunner, Hive_test e Beetest.
- **Amostragem (Sampling)**: permite que um subconjunto de dados seja analisado sem a necessidade de análise de todo o conjunto.

    - Com uma amostra representativa, uma consulta pode retornar resultados significativos muito mais rapidamente, consumindo menos recursos.
    - O Hive oferece o TABLESAMPLE, que permite que tabelas sejam experimentadas, em vários níveis de granularidade - retornando subconjuntos ou blocos HDFS ou apenas os primeiros n registros de cada input split.
    - Uma UDF pode ser implementada para filtrar os registros segundo um algoritmo de amostragem.

Detalhes do Projeto Apache Hive

Apache Hive foi desenvolvido predominantemente em Java. O HiveQL é a linguagem baseada em SQL do HIVE.
Ela "aplica" a sintaxe SQL na criação de tabelas, carga de dados e consulta.
Também permite a incorporação de scripts personalizados de MapReduce.
Esses scripts podem ser escritos em qualquer idioma usando uma interface streaming simples (leitura das linhas na entrada padrão e gravação na saída padrão).

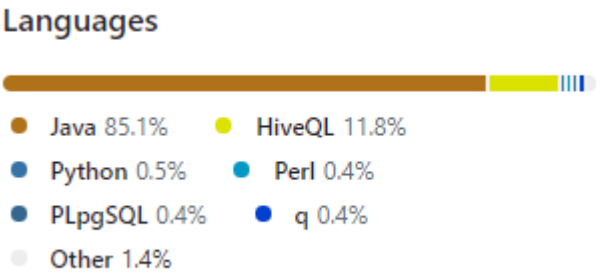


Figura 6 - Linguagens do Hive

TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.



> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **4.0.0**, distribuída pelo Helm Chart `tdp-hive-metastore` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- Apache Hive - cwiki
- Apache Hive.org
- GitHub - Apache Hive

© Copyright Tecnisys 2026. All rights reserved.

Apache Flink

Motores de Computação Distribuída



O conceito de computação distribuída não é novo e tornou-se ainda mais conhecido com a "explosão" de dados dos últimos anos, e o crescimento da necessidade de tratar grandes fluxos de informação com capacidade preditiva.

Entre os seus principais benefícios podemos citar a capacidade de adicionar mais nós de processamento (escalabilidade), na proporção do crescimento dos dados, sem comprometer performance ou disponibilidade dos sistemas e a capacidade de processamento em tempo real - a distribuição dos trabalhos em diversos nós agiliza e proporciona eficiência à tomada de decisão.

Na era do Big Data, surgiram vários modelos de programação e frameworks para execução de *jobs* em lote de forma otimizada e distribuída, como o Map Reduce, e outros, com a capacidade de processamento de dados em larga escala, em paralelo e com tolerância a falhas, muito úteis para ETL (Extract, Transform, Load), como o Apache Hadoop, Apache Spark, Apache Beam e Apache Flink.

Outra forma de processar dados decorrente da era Big Data é o *streaming*, um tipo de *engine* de processamento de dados projetado para tratar *datasets* infinitos (os dados surgem infinitamente e não há garantia da ordem de chegada). Dentre os principais *frameworks/engines* de processamento distribuídos codificados para *streaming*, citamos o *Apache Flink*, *Apache Storm*, *Apache Flume* e *Apache Samza*. Estes mecanismos recebem mensagens através da leitura de um sistema de mensageria (*source*) - como o *Apache Kafka* - processa-os em tempo real, filtrando os que contêm os dados procurados e envia-os para uma saída.

Apache Flink

© Copyright Tecnisys 2026. All rights reserved.



O *Apache Flink* é uma estrutura e mecanismo de processamento distribuído para computação com estado (funções que assumem um estado e retornam um novo estado) de *streams* de dados ilimitados e limitados. Foi criado pela equipe de pesquisa da Berlin Technical University, liderada por Stephan Ewen. Começou como um projeto acadêmico chamado Stratosphere, por volta de 2009. Em 2014, o projeto foi incorporado à Apache Software Foundation, tornando-se um projeto de incubação. Posteriormente foi promovido a um projeto de nível superior da Apache. Desde então, o Flink evoluiu significativamente, tornando-se uma das principais plataformas de processamento de fluxo de dados em tempo real.

Características do Apache Flink

O Apache Flink suporta múltiplas noções de tempo para processamento de fluxos baseado em estado.

Dentre suas principais características, destacamos:

- **Processamento de estados**: Gerencia estados complexos, mesmo em operações de grande escala.
- **Correção e consistência**: Oferece garantias de exatidão de dados (exatidão-uma-vez).
- **Processamento de tempo de evento**: Permite lidar com dados em tempo real, considerando a hora em que eventos ocorrem.
- ***APIs* em camadas**: Abordagem modular e flexível para construir aplicações. Fornece várias *APIs* em diferentes níveis de abstração, incluindo *SQL*, DataStream e DataSet:
  - ***API* de Processamento de Fluxo de Dados (*DataStream API*)**: Usada para aplicações de processamento de fluxo de dados em tempo real.
  - ***API* de Conjunto de Dados (DataSet API)**: Voltada para processamento de dados em lote.
  - ***API SQL* e *Table***: Permite aos usuários escrever consultas em *SQL* ou em uma linguagem de tabela similar para manipulação de dados.

  Essas *APIs* permitem aos desenvolvedores escolher o nível de abstração que melhor se adapta às suas necessidades.

© Copyright Tecnisys 2026. All rights reserved.

Arquitetura do Apache Flink

A arquitetura do Apache Flink é composta por vários componentes:

- ***JobManager***: Coordena a distribuição de tarefas e a gestão de recursos. Ele também lida com o checkpointing para garantir a tolerância a falhas.
- ***TaskManager***: Executa as tarefas (ou *jobs*) efetivamente. Cada TaskManager pode executar múltiplas tarefas simultaneamente.
- ***Dispatcher***: Oferece uma interface REST para submeter aplicações Flink e iniciar um novo JobMaster para cada trabalho, além de executar o *WebUI* do Flink.
- **Cliente Flink**: Prepara e envia um fluxo de dados para o JobManager. Pode desconectar ou permanecer conectado para receber relatórios de progresso.
- ***Resource Manager***: Responsável pela alocação de recursos e provisão no *cluster* Flink, gerenciando slots de tarefas.
- **JobMaster**: Gerencia a execução de um único *JobGraph*.
- **Fontes e Destinos de Dados (*Sources* and *Sinks*)**: Conectam-se com sistemas de armazenamento externos para entrada e saída de dados.

Esta arquitetura permite o processamento distribuído e escalável de grandes volumes de dados. Para mais informações detalhadas, visite a documentação oficial do Apache Flink em Apache Flink Architecture.

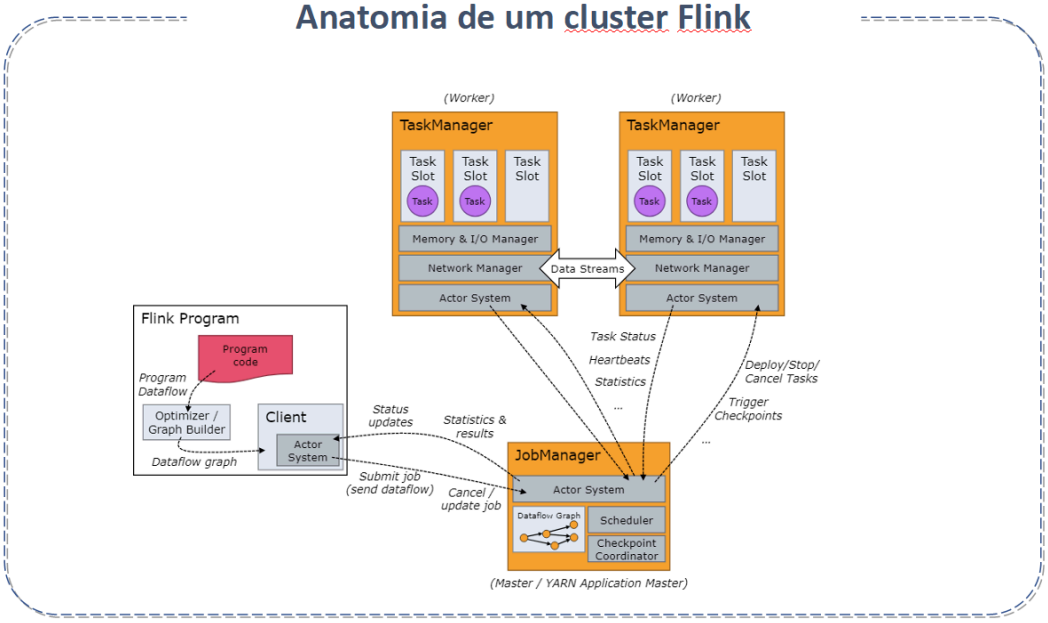


Figura 1 - Anatomia de um _cluster_ Flink

Recursos do Apache Flink

© Copyright Tecnisys 2026. All rights reserved.



- **Análise de fluxo e lote**: Suporta tanto processamento em tempo real quanto em lote.
- **Aplicações orientadas a eventos**: Ideal para sistemas que reagem a eventos em tempo real.
- **Alta performance**: Projetado para ser rápido, com baixa latência e alto throughput.

Quando usar o Apache Flink

- O Apache Flink é Ideal para aplicações que exigem processamento de fluxo de dados em tempo real e análise de eventos.
- É Recomendado para cenários com requisitos de tolerância a falhas e garantias de consistência de dados.
- É Adequado para sistemas que necessitam de processamento de dados em grande escala e com baixa latência.

Quando o Apache Flink não se aplica

- O Apache Flink não é a melhor opção para projetos simples de manipulação de dados onde soluções mais leves, como scripts ou ferramentas de *ETL* tradicionais, seriam suficientes.
- Pode ser excessivo para aplicações que não requerem recursos avançados de processamento de fluxo de dados ou análise em tempo real.
- É menos eficiente para tarefas onde o processamento em lote, sem a necessidade de tratamento em tempo real, é suficiente.

Principais diferenças entre Apache Spark e Apache Flink

- **Modelo de Processamento**: Uma das principais diferenças entre as duas ferramentas é o modelo de processamento. O Spark usa o modelo de processamento de dados RDD (*Resilient Distributed Datasets*) - uma coleção distribuída e imutável de objetos. O Apache Flink usa o modelo de processamento de dados baseado em fluxos - uma coleção de eventos processados em tempo real à medida que chegam.
- **Velocidade**: O Apache Flink geralmente é mais rápido que o Apache Spark para processamento de dados em tempo real, isto porque seu modelo de processamento elimina a necessidade de transformar dados em RDDs antes de processa-los.
- **Tolerância a falhas**: Ambas as ferramentas são tolerantes a falhas, mas Spark usa um mecanismo de *checkpointing* baseado em disco para manter a integridade dos

© Copyright Tecnisys 2026. All rights reserved.

dados e Flink usa uma abordagem baseada em memória, armazenando os dados em cache, o que permite uma recuperação mais rápida no caso de falhas.
- **Suporte a eventos complexos**: O Apache Flink oferece suporte nativo a processamento de eventos complexos, o que permite a análise de padrões e correlações em fluxos de dados em tempo real, ao contrário do Spark que requer bibliotecas adicionais e ajustes de configuração.

Em suma, o Apache Spark é frequentemente usado para processamento de dados em lote e aplicações que não exigem uma latência muito baixa, como análise de *logs* e processamento de dados históricos. O *Apache Flink* é frequentemente usado para processamento de dados em tempo real, como análise de fraudes e monitoramento de IoT.

Principais diferenças entre *Apache Storm* e *Apache Flink*

A comparação entre *Apache Flink* e *Apache Storm* inclui vários aspectos:

- **Uso**: *Flink* é mais para processamento unificado de lote e fluxos, enquanto Storm é focado em processamento de fluxo em tempo real.
- **Processamento de Dados**: *Flink* oferece processamento de dados em lote e em fluxo; *Storm* é ótimo para processamento de fluxo.
- **Transformação de Dados**: *Flink* tem opções ricas para transformação de dados; *Storm* é mais limitado.
- **Suporte a Aprendizado de Máquina**: *Flink* suporta o aprendizado de Máquina, enquanto o *Storm* não.
- **Linguagem de Consulta**: *Flink* usa uma linguagem semelhante ao SQL; *Storm* não tem linguagem de consulta nativa.
- **Modelo de Implantação**: Ambos oferecem diferentes opções de implantação.
- **Integração com Outros Serviços**: Ambos se integram bem com outros serviços.
- **Escalabilidade**: *Apache Storm* destaca-se em alta escalabilidade.
- **Desempenho e Confiabilidade**: Ambos têm bom desempenho e são confiáveis.

Boas Práticas para Apache Flink_Apache Flink_

- **Gerenciamento Eficiente de Estado**: Estruture e gerencie estados de forma eficiente para otimizar desempenho e garantir a consistência dos dados.
- **Otimização de Desempenho**: Monitore métricas de desempenho, faça ajustes finos em configurações e utilize estratégias de particionamento de dados para maximizar a eficiência.

© Copyright Tecnisys 2026. All rights reserved.



- **Estratégias de Recuperação de Falhas**: Implemente estratégias robustas de *checkpointing* e recuperação para garantir a tolerância a falhas.
- **Escalabilidade e Balanceamento de Carga**: Desenvolva aplicações com escalabilidade em mente, garantindo um balanceamento efetivo da carga entre os nodos do *cluster*.
- **Testes e Validação**: Realize testes extensivos para garantir a confiabilidade e a precisão das aplicações Flink em diferentes cenários.
- **Atualizações e Manutenção**: Mantenha-se atualizado com as últimas versões e práticas recomendadas do Flink para aproveitar melhorias e correções.

Detalhes do Projeto Apache Flink

O Apache Flink é predominantemente utilizado com a linguagem de programação Java. Também oferece suporte a *Scala*, uma vez que a sua *API Scala* se integra perfeitamente com a *API* Java. Além disso, para operações de consultas de dados, o Flink suporta *SQL* através de sua API de *Table* e *SQL*. Essa flexibilidade permite que os desenvolvedores escolham a linguagem que melhor se adapta às suas necessidades e preferências.

Fontes:

- Apache Flink

© Copyright Tecnisys 2026. All rights reserved.

Great Expectations

Validação de Dados e Garantia de Qualidade



> **Recurso disponível somente a partir da versão 2.3.**

No contexto do Big Data e dos pipelines de dados, a validação de dados desempenha um papel essencial.

Trata-se de um processo cuidadosamente planejado para assegurar que os dados coletados ou processados atendam a padrões, critérios e formatos previamente definidos.

Antes de avançarem para análises, relatórios ou modelos de machine learning, os dados passam por um rigoroso "pente fino", garantindo sua qualidade e consistência. Essa validação é um ponto de verificação crucial, pois dados incorretos podem levar a decisões desastrosas, análises equivocadas e, no caso de organizações reguladas, até mesmo implicações legais.

A importância desse processo vai além da simples identificação de erros. Ele é o alicerce que garante a confiança organizacional em seus dados.

Imagine confiar em dados que não foram verificados — erros simples como valores ausentes ou inconsistências entre tabelas podem comprometer resultados inteiros. Além disso, a automação do processo de validação reduz a intervenção manual, tornando-o mais eficiente e prevenindo gargalos em pipelines complexos. Quando feito corretamente, ele também garante conformidade com requisitos regulatórios e políticas internas, indispensáveis em setores como saúde, finanças e tecnologia.

© Copyright Tecnisys 2026. All rights reserved.



Figura 1 - Garantia da Qualidade

Características do Great Expectations

O Great Expectations (GX) é uma ferramenta de código aberto projetada para tornar todo o processo de validação e garantia de qualidade de dados mais acessível, automatizado e eficiente. Ele resolve desafios enfrentados por equipes de ciência de dados e engenharia ao lidar com pipelines complexos, como falta de visibilidade sobre a qualidade dos dados e dificuldades para integrar verificações contínuas nos fluxos de trabalho.

No coração do Great Expectations está o conceito de "expectativas". Essas expectativas são, essencialmente, regras que definem como os dados devem se comportar. Elas podem ser simples, como "esta coluna não deve conter valores nulos", ou sofisticadas, como "os valores desta coluna devem estar em um intervalo entre 0 e 100, com uma média esperada de 50".

A flexibilidade para definir regras específicas torna o GX uma ferramenta poderosa para atender às diversas necessidades de validação de dados.

Uma das características mais marcantes da ferramenta é sua capacidade de gerar relatórios interativos, conhecidos como Data Docs. Esses relatórios não apenas documentam a qualidade dos dados, mas também permitem uma análise visual das validações realizadas, facilitando a comunicação entre equipes técnicas e não técnicas. Eles são ideais para promover transparência em organizações onde a qualidade dos dados é crítica.

© Copyright Tecnisys 2026. All rights reserved.

Arquitetura do Great Expectations

A arquitetura do Great Expectations foi projetada para ser modular e adaptável. Ela é composta por três elementos principais:

- **Expectations (Expectativas)**: Representam as regras que os dados precisam cumprir. Por exemplo, "os valores da coluna 'idade' devem estar entre 0 e 120".
- **Validation Results (Resultados de Validação)**: Indicam se os dados atendem ou não às expectativas, oferecendo métricas detalhadas sobre conformidade.
- **Data Docs (Documentação de Dados)**: São relatórios interativos gerados automaticamente que fornecem uma visão clara e acessível da qualidade dos dados validados.

Esses componentes trabalham juntos para criar um fluxo de trabalho que começa com a definição de expectativas, passa pela validação dos dados e culmina na geração de relatórios que documentam e monitoram a qualidade dos dados ao longo do tempo.

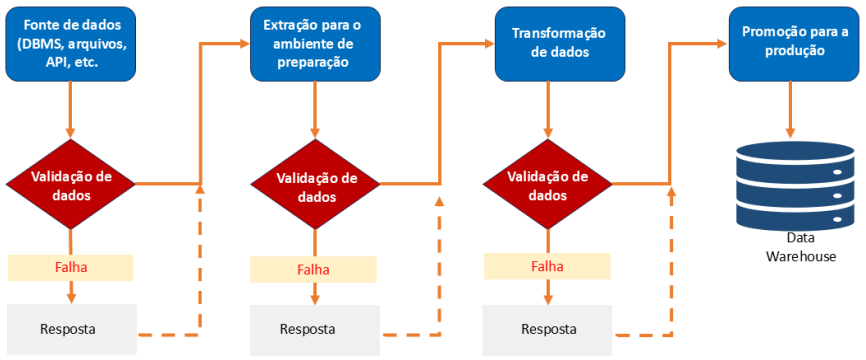


Figura 2 - Great Expectations - Validação de Dados

Filosofia do Great Expectations

A filosofia do Great Expectations é simples, mas transformadora: permitir que as organizações definam e validem o que esperam de seus dados de maneira estruturada e reutilizável.

Ele formaliza essas expectativas em um formato configurável, frequentemente representado em JSON, permitindo fácil manutenção e integração com outras

© Copyright Tecnisys 2026. All rights reserved.

ferramentas.

Com um catálogo de mais de 70 tipos de expectativas predefinidas, o GX cobre a maioria das necessidades de validação de dados.

Além disso, é possível criar expectativas personalizadas para casos específicos, garantindo flexibilidade e escalabilidade.

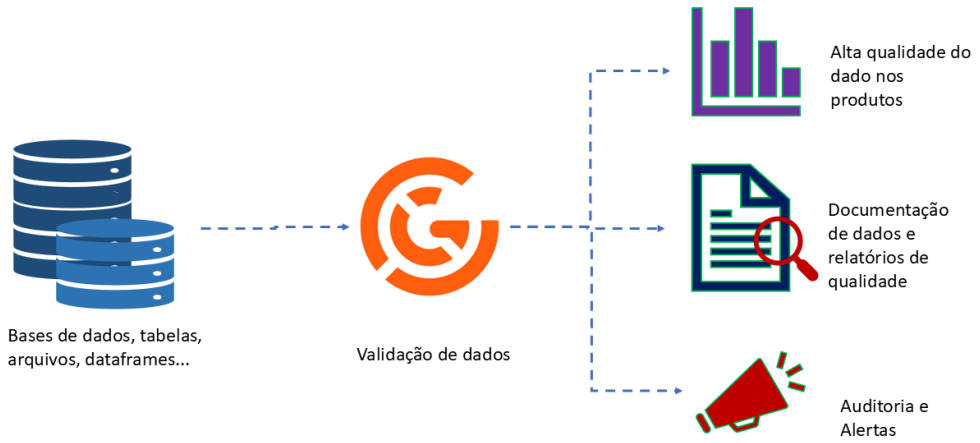


Figura 3 - Modelo do Great Expectations

Recursos do Great Expectations

O Great Expectations oferece uma série de recursos que o destacam como uma solução robusta para validação de dados:

- **Amplo suporte a Fontes de Dados*: Compatível com arquivos CSV, Parquet, bancos de dados relacionais e data lakes.
- **Execução Local ou em Nuvem*: Flexibilidade para rodar em diferentes ambientes.
- **Configuração simples e intuitiva*: Interfaces amigáveis para definir e gerenciar expectativas.
- **Comunidade ativa*: Constantes atualizações e suporte da comunidade.

O Great Expectations é mais do que uma ferramenta de validação; é um marco na evolução da garantia de qualidade de dados, oferecendo soluções claras para desafios complexos.

© Copyright Tecnisys 2026. All rights reserved.

Detalhes do Projeto Great Expectations

O Great Expectations foi desenvolvido em Python, aproveitando a popularidade e versatilidade da linguagem para facilitar sua integração em pipelines de dados. Python é amplamente utilizado por cientistas de dados e engenheiros, tornando o GX acessível e fácil de adotar.

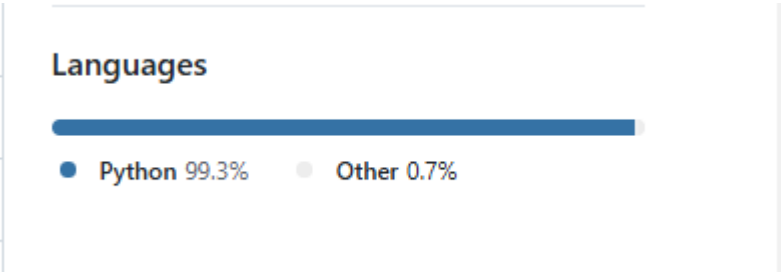


Figura 4 - Linguagem do Great Expectations

Fontes:

- Site Great Expectations

© Copyright Tecnisys 2026. All rights reserved.

Apache HBase



Base de dados NoSQL

O Modelo NoSQL (*no SQL* ou *not only SQL*) representa bancos de dados não relacionais (ou não somente relacionais, conforme definições mais recentes). Trata-se de uma classe de banco de dados que fornece mecanismos de armazenamento e recuperação de dados modelados de formas diferentes das relações tabulares usadas pelos bancos de dados relacionais.

Estes bancos de dados existem desde a década de 1960, mas alcançaram popularidade na década de 2000, desencadeada pelas necessidades de empresas como Facebook, Google e Amazon.com. São cada vez mais usados em Big Data e aplicações Web em tempo real.

Os Bancos de dados NoSQL podem admitir linguagens de consulta similares à SQL.

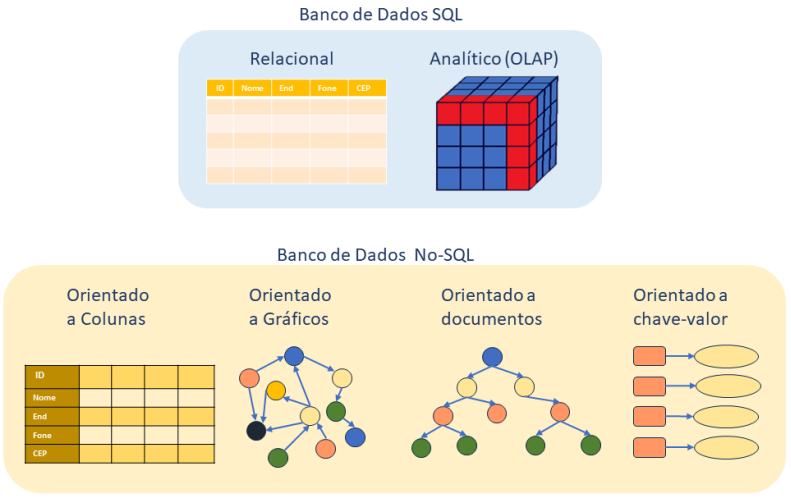


Figura 1 - Modelos de Banco de Dados

Características do Apache HBase

© Copyright Tecnisys 2026. All rights reserved.

HBase é um banco de dados distribuído, de código aberto, não-relacional ("NoSQL"), orientado a colunas, que executa sobre o HDFS, criado para fornecer acesso aleatório eficiente e leitura/gravação em tempo real em grandes conjuntos de dados distribuídos.

Foi criado pela empresa Powerset para hospedar tabelas muito grandes e, posteriormente, absorvido pela Fundação Apache como parte do Projeto Hadoop, tornando-se uma ótima opção para combinar e armazenar dados multi-estruturados e esparsos.

O HBase está integrado nativamente com o Hadoop e funciona perfeitamente ao lado de outros "motores" de acesso a dados, através do YARN. Possui muitos recursos que suportam escalabilidade linear e modular.

Seus *Clusters* se expandem pela adição de *RegionServers* que são hospedados em servidores "commodity class" (hardware comum). Se um *Cluster* se expande de 10 para 20 RegionServers, por exemplo, ele dobra sua capacidade de armazenamento e processamento.

Suas principais características incluem:

- **Leitura e gravação consistentes**: O HBase é um *DataStore* consistente, o que o torna muito adequado para tarefas como *high speed counter aggregation*.
- **Fragmentação" automática**: As tabelas HBase são distribuídas no *Cluster* via Regions(regiões), e estas regiões são automaticamente divididas e redistribuidas à medida que os dados "crescem".
- **Tolerância à falhas automática do RegionServer**: Os dados de entrada "residem" em regiões disponibilizadas pelos *RegionServers* . Se um *RegionServer* falhar, o servidor mestre designará outro para assumir as regiões que foram tratadas pelo *RegionServer* com falha.
- **Integração Hadoop/HDFS**: HBase suporta HDFS nativamente como seu sistema de arquivos distribuídos.
- **Suporte ao MapReduce::**: O HBase oferece suporte a processamento fortemente paralelizado via MapReduce, atuando tanto como fonte quanto como coletor.
- **API Java Client:** HBase suporta API Java "easy to use" para acesso programático.

© Copyright Tecnisys 2026. All rights reserved.

- **API Thrift/REST::** HBase suporta *Thrift* e *REST* para "front-ends" não-Java.
- ***Block-cache* e *Bloom Filters*:** HBase oferece suporte para *high volume query otimization* .
- **Gerenciamento Operacional:** HBase provê Web pages integradas para percepção operacional e métricas JMX.
- **Alta disponibilidade:** HBase provê alta disponibilidade com:
    - Informações de topologia de *Cluster* altamente disponíveis através de implantações de produção com múltiplas instâncias *HMaster* e *Zookeeper*.
    - Distribuição de dados em vários nós garantindo que a perda de um único nó afete somente os dados armazenados nesse nó.
    - HBase HA permite o armazenamento de dados, garantindo que a perda de um único nó não resulte na perda de disponibilidade de dados.
    - O formato *HFile* armazena dados diretamente no HDFS e pode ser lido ou escrito por Apache Hive, Apache PIg, MapReduce e Apache Tez, favorecendo análises profundas no HBase sem movimentação de dados.

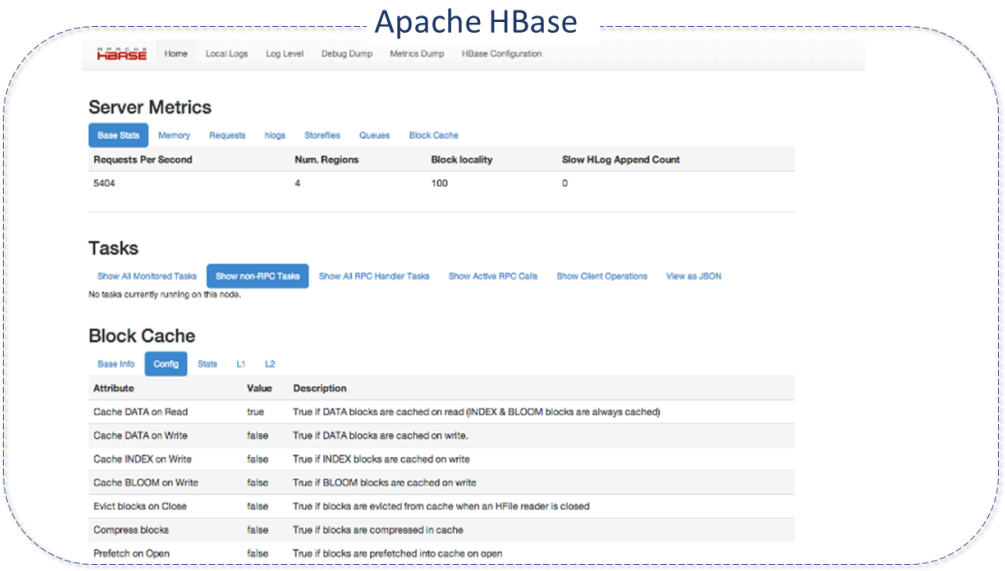


Figura 2 - Interface HBase

Arquitetura do Apache HBase

A arquitetura do HBase também segue o modelo mestre/escravo e é formada por três componentes principais:

© Copyright Tecnisys 2026. All rights reserved.

- **HMaster (Servidor mestre):**: é um processo responsável por designar regiões para Region Servers no *Cluster* Hadoop para balanceamento de carga. É responsável pela monitoração de todas as instâncias de Region Server no *Cluster* HBase, e age como uma interface para todas as mudanças de metadados. O *master* tipicamente é executado em um servidor dedicado, dentro do *cluster* Hadoop.
  Existem outros *masters* em *standby* disponíveis para substituição em caso de indisponibilidade e o Zookeeper cuida disso.

  > **NOTE**
  >
  > É possível que o master fique inativo por algum tempo e é possível o *Cluster* Hadoop trabalhar sem ele durante este período, mas é sempre bom reativar o master o mais breve possível, porque ele conduz algumas funcionalidades criticas como:
  >
  > - Gerenciamento e monitoramento do *Cluster* Hadoop;
  > - Controle do *failover*
  > - Atribuição de regiões aos *Region Servers* , com apoio do Zookeeper.
  > - Balanceamento de carga das regiões entre os *Region Servers*.
  > - Alterações de esquema e outras operações de Metadados.
  > - Operações DDL, como criação e exclusão de tabelas.

- **Regionservers:** mantêm as múltiplas regiões designadas pelo HMaster. Várias regiões são combinadas em um único RegionServer.

  São responsáveis pela comunicação cliente e por executar todas as operações relacionadas a dados. Qualquer requisição de leitura ou gravação para as regiões são conduzidas pelo Region Server.

  O *Region Server* é implementado pelo *HRegionServer* e executa em todo nó de dados no *Cluster* HBase. É composto pelos seguintes componentes:

  - *Write-Ahead Log (WAL):* O WAL(registro de gravação antecipada) registra todas as alterações de dados do HBase. Em uma situação normal, ele não é necessário, pois as mudanças de dados são movidas do MemStore para os StoreFiles Entretanto, se um *RegionServer* travar ou ficar indisponível antes que o MemStore seja liberado, o WAL garante que as mudanças possam ser repetidas. Se a gravação no WAL falhar, toda a operação falha.

© Copyright Tecnisys 2026. All rights reserved.



O WAL é anexado a cada *Region Server*. Usualmente, existe apenas uma única instância de WAL por *Region Server*. A exceção é o *RegionServer* que carrega a tabela *HBase:meta*. A tabela tem seu próprio WAL dedicado.

- *Block Cache::* *Block Cache* reside no *Region Server* e é responsável pelo armazenamento de dados acessados com frequência, na memória, auxiliando no incremento da performance. O *Block Cache* segue o conceito *least recently used(LRU)*, em que registros menos utilizados recentemente serão removidos.

- *MemStore:* É um cache de gravação. Uma memória temporária para todos os dados de entrada. Responsável por armazenar dados ainda não gravados no disco. Existem múltiplos *MemStores* no *Cluster* HBase.

- *HFile:* São os arquivos ou células nos quais são armazenados os dados. Quando o *MemStore* está cheio, grava dados no *HFile*, no disco.

- *Regions:* São o elemento basico de disponibilidade e distribuição de tabelas. São compostos por uma *Column Family*. Tabelas HBase são divididas por intervalos *row key* nas regiões. Todas as linhas entre *region start key* e *region end key* estarão disponíveis na região. Cada região é designada para um *region server* , que gerencia as solicitações *leitura/gravação* para esta região.
  Como regiões são blocos básicos de escalabilidade no HBase, é recomendado um baixo número de regiões por *Region Server* para garantir alta performance. A arquitetura HBase usa um processo de fragmentação automática para manter os dados. Neste processo, sempre que uma tabela HBase fica muito longa, é distribuída pelo sistema com o auxílio do HMaster.

- *Zookeeper:* Zookeeper realiza a coordenação distribuída, provendo regiões para os Region Servers e também recuperando regiões, no caso de falha do Region Server, carregando-as em outros Region Servers. Se um cliente deseja compartilhar ou permutar com regiões, precisa abordar o Zookeeper antes. O serviço HMaster e todos os *Region Servers* são registrados com o serviço ZooKeeper. O Cliente acessa o ZooKeeper para conectar-se com *Region Servers* e *HMaster*. Em caso de falha em um nó do *Cluster* HBase, o ZKquoram do Zookeeper ativará mensagem de erro e iniciará o *repaire failed nodes*.
  O serviço ZooKeeper mantém e acompanha todos os *Region Servers* no *Cluster* HBase e coleta informações como número de *Region Servers* , datanodes designados, etc. É também responsável por estabelecer a comunicação cliente com *Region Servers* , mantendo o controle de falhas de servidor e partições de rede, informações de configuração, etc.

© Copyright Tecnisys 2026. All rights reserved.

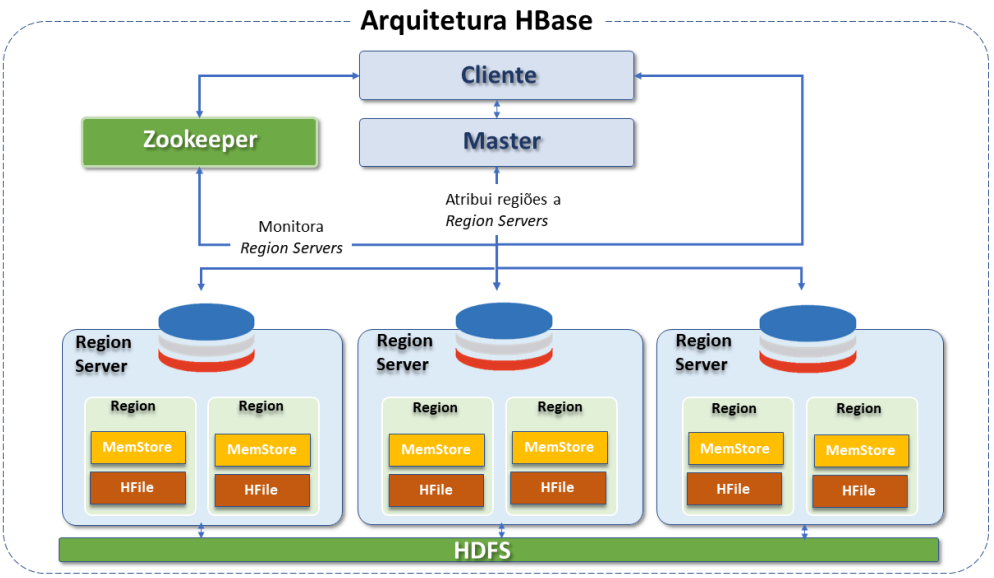


Figura 3 - Arquitetura HBase

Como o Apache HBase funciona

HBase escala linearmente, exigindo que todas as tabelas tenham uma chave primária. O espaço da chave está dividido em blocos sequenciais que são, então, atribuídos a uma região.

Os RegionServers possuem uma ou mais regiões, de modo que a carga seja distribuída uniformemente em todo o *Cluster*. Se as chaves dentro de uma região são frequentemente acessadas, o HBase pode subdividir a região de modo que o corte manual de dados não seja necessário.

Os servidores do Zookeeper e HMaster disponibilizam informações sobre a topologia de *Cluster* aos clientes. Os clientes se conectam a estes e baixam:

- a lista de RegionServers,
- as regiões contidas nesses RegionServers, e
- os intervalos de chaves hospedados pelas regiões.

Como os clientes sabem exatamente onde está cada informação no HBase, podem entrar em contato diretamente com o RegionServer sem necessidade de um "coordenador central".

Regionserver inclui uma *memstore* para armazenar em cache as linhas frequentemente acessadas na memória.

© Copyright Tecnisys 2026. All rights reserved.



Modelo de Dados HBase

No HBase os dados são armazenados em tabelas, que possuem linhas e colunas, podendo, sua estrutura, ser entendida como um mapa multidimensional.

Seus principais componentes são:

- **Tabelas:** Como a arquitetura HBase é orientada a colunas, os dados são armazenados em tabelas que estão no formato tabular. Estas tabelas são declaradas antecipadamente no momento da definição do "esquema". Uma tabela consiste em várias linhas.

- **Rowkey:** Uma linha no HBase corresponde a uma *Rowkey* e uma ou mais colunas com valores associadas a ela. As linhas são classificadas alfabeticamente pela *RowKey* à medida em que são armazenadas.

- **Colunas:** Uma coluna no HBase consiste em uma família de colunas (column_family) e um qualificador de coluna delimitados por um caractere ":" (dois pontos). São os diferentes atributos do *dataset*. Cada *Rowkey* pode ter um número ilimitado de colunas.

    - As colunas não são definidas previamente na estrutura, como no Banco de Dados Relacional.
    - Não é obrigatório que todas as colunas estejam presentes em todas as linhas.
    - As colunas não armazenam valores nulos.
    - Todas as colunas de uma *Column Family* são armazenadas juntas num arquivo (HFile) juntamente com a *Rowkey*.
- **Column Family:** São o agrupamento de várias colunas. Uma simples requisição de leitura a uma *column family* dá acesso a todas as colunas daquela familia, tornando rápida e fácil a leitura do dado. Uma tabela deve ter pelo menos uma *Column Family*, que é criada com a tabela.

- **Qualificadores de coluna (*Column Qualifiers*):** São como títulos das colunas ou nomes de atributos em uma tabela normal. Um qualificador de colunas é adicionado a uma *column family* para fornecer o índice para um determinado dado.

- **Célula:** É uma combinação entre linha, *column family* e qualificador de coluna e contém um valor e um _timestamp, que representa a versão do valor.

© Copyright Tecnisys 2026. All rights reserved.



- **Timestamp:** O dado é armazenado no modelo de dados HBase com um Timestamp. O *Timestamp* é escrito ao lado de cada valor e é o identificador para uma determinada versão do valor.

Namespace

Namespace consiste de um agrupamento lógico de tabelas análogas a um Banco de Dados em sistemas de SGBD relacionais. Essa abstração estabelece as bases para recursos de multilocação:

- **Gestão de cotas:** Restringindo a quantidade de recursos (regiões, tabelas) que um namespace pode consumir.
- **Gestão de segurança:** Fornecendo outro nível de gestão de segurança para "locatários".
- **Grupos de Region Servers:** Uma tabela/namespace pode ser fixado(a) em um subset de RegionServers, garantindo um certo nivel de isolamento.
- **Gestão da Segurança do Namespace:** Um Namespace/table pode ser fixado em um subconjunto de RegionServers, garantindo isolamento.

Operações do Modelo de Dados HBase

Como todo database, o HBase suporta um conjunto de operações básicas referenciadas como CRUD (Criação, Exclusão, leitura e atualização).

Versões

Uma tupla (linha, coluna, versão) especifica uma célula. É possível haver um número de células ilimitado, onde a linha e colunas são as mesmas mas o endereço da célula difere em sua dimensão *versão*.

Joins

HBase não suporta *joins* (junções) na forma como os RDBMS. As leituras do HBase são *Get* e *Scan*. Para viabilizar o *join*, existem duas estratégias:

- Desnormalizar os dados ao gravar no Hbase.
- Ter tabelas de pesquisa e fazer a *junção* entre as tabelas *HBase* e um aplicativo ou código *MapReduce*.

© Copyright Tecnisys 2026. All rights reserved.



A melhor estratégia depende do que se deseja fazer.

HBase e Design do *Schema*

Projetar um *Schema* para Bigtable é diferente de projetar um *schema* para o banco de dados relacional. No Bigtable, um esquema é um *blueprint* ou modelo de uma tabela, incluindo a estrutura dos componentes da tabela.

No Bigtable, um esquema é um *blueprint* ou modelo de uma tabela, incluindo a estrutura dos componentes da tabela a seguir:

- Rowkeys
- Grupos de Colunas (incluindo as políticas de coleta de lixo)
- Colunas

Os principais conceitos que se aplicam ao design do *Schema* em Bigtable são:

- Bigtable é uma armazenamento de chave/valor, não um armazenametno relacional. Ele não é compativel com "mesclagens" e as transações são compatíveis apenas com uma única linha.
- Cada tabela possui apenas um índice: a *rowkey*. Não há indices secundários. Cada *rowkey* precisa ser única.
- As linhas são classificadas lexicograficamente por *rowkey*, da *string* que tem menos bytes para a que tem mais.
- Os grupos de colunas não são armazenados em alguma ordem específica.
- As colunas são reunidas por grupo e classificadas em ordem lexicográfica dentro desse grupo.
- A intersecção de uma linha e coluna pode contér várias células com *timestamp* de data/hora. Cada célula contém uma versão exclusiva e com *timestamp* dos dados para essa linha e coluna.
- Todas as operações são atômicas no nível da linha. Uma operação afeta uma linha inteira ou nenhuma linha.
- O ideal é leituras e gravações distribuídas por igual pelo espaço da linha de uma tabela.
- As tabelas do *Bigtable* são esparsas. Uma coluna não ocupa espaço em uma linha que não usa a coluna.

A comunidade Hbase indica, para tratar este assunto, o site práticas e recomendações de criação de esquema para maiores detalhes.

© Copyright Tecnisys 2026. All rights reserved.

Recursos do Apache HBase

- **Segurança no Apache HBase:**
    - **Segurança da Interface para o Usuário Web:** HBase provê mecanismos para assegurar vários componentes e aspectos e no seu relacionamento com a infraestrutura Hadoop, bem como clientes e recursos externos ao Hadoop.
    - **Segurança do acesso pelo cliente ao Apache HBase:** As versões mais recentes do Apache HBase suportam autenticação opcional SASL de clientes. A comunidade recomenda a leitura do artigoUnderstanding User Authentication and Authorization in Apache HBase: para maiores detalhes.
    - **Segurança no nível de transporte:** A partir da versão 2.6.0 o HBase suporta criptografia TLS na comunicação entre o cliente-servidor e *Master-RegionServer*. TLS é um protocolo criptografico padrão desenhado para prover segurança de comunicações entre redes de computadores.
    - **Segurança no acesso ao HDFS e Zookeeper:** O HBase requer Zookeeper e HDFS seguros impedir o acesso ou modificação de metadados e dados. HBase use o HDFS para manter seus arquivos de dados, *write ahead logs*(WALs) e outros dados. E usa Zookeeper para armazenar algums metadados para operações (endereço master, lock de tabelas, recuperações, etc.)
    - **Segurança no acesso aos dados do cliente:** A segurança dos dados do cliente também deve ser considerada. Algumas estratégias são oferecidas para isto:
        - Role-based Access Control (RBAC - controle de Acesso baseado em papéis): Onde usuários e grupos podem ler e gravar em um recurso específico ou executar um *endpoint coprocessor*, usando o paradigma de papéis.
        - Labels de visibilidade que viabilizam a rotulação de células e controle de acesso às células rotuladas, visando restringir quem pode ler e gravar em determinado subconjunto de dados.
        - Criptografia transparente de dados em repouso no sistema de arquivos subjacente, tanto HFiles quanto Wal, protegendo dados em repouso de uma invasão.

© Copyright Tecnisys 2026. All rights reserved.



- **Inmemory compaction (Compactação na memória):** A compactação na memória (Aka Acordion) é um novo recurso do HBase - 2.0.0.
- **RegionServer Offheap Read-Write Path (Caminho de leitura e gravação "fora da pilha" RegionServer):** Para ajudar a reduzir as latências de RPC P99/P999, o Hbase 2.x fez com que o caminho de leitura e gravação usasse um pool de *buffers offheap*.
- ***Backup* e *Restore*:** O *Backup* e *Restore* do HBase provê a habilidade de criar *backups full* e *incremental* em tabelas do *Cluster* HBase.
- **Replicação síncrona:** A Replicação A replicação no HBase é assincrona. Portanto, se o *Cluster* Master falhar, o *Cluster* escravo pode não ter os dados mais atuais. Para garantir consistência o usuário não poderá alternar para o *Cluster* escravo.
- **APIs:** Além das APIs nativas, o HBase suporta APIs externas.
- **Coprocessors:**(Co-processadores) O framework de Co-processadores fornecem mecanismos para executar códigos customizados diretamente nos RegionServers que gerenciam o dado.

Quando usar o Apache HBase

O Apache HBase não é adequado para todas as situações.

- É um bom candidato apenas para situações com um grande volume de dados. Para pequenos e médios volumes de dados, os RDBMS tradicionais podem ser melhores, pois seus dados podem se concentrar em um ou dois nós.
- Não possui tantos recursos extras quantos os RDBMS (colunas digitadas, índices secundários, transações, linguagens avançadas de consulta, etc.).
- Necessita de estrutura de Hardware adequada. Mesmo o HDFS não funciona bem com menos que 5 DataNodes e 1 NameNode.

Diferença entre Apache HBase e HDFS

HDFS é um sistema de arquivos distribuídos adequado para armazenamento de grandes arquivos. Entretanto, não é um sistema de arquivos de uso geral e não fornece pesquisas rápidas de registros individuais em arquivos.

© Copyright Tecnisys 2026. All rights reserved.

O HBase, por sua vez, é construído sobre o HDFS e fornece pesquisas rápidas de registros (e atualizações) para grandes tabelas.

O HBase, internamente, dispõe seus dados em *StoreFiles* indexados do HDFS para pesquisas de alta velocidade.

Boas Práticas para o Apache HBase

- **Hotspotting:** Como as linhas do HBase são classificada lexigraficamente por chave de linha (processo que otimiza as varreduras, permitindo que sejam armazenadas linhas relacionadas próximas umas das outras), quando mal projetadas, as chaves de linha podem ser uma fonte de *hotspotting* (quando uma grande quantidade de tráfego do cliente é direcionada para um nó ou apenas alguns nós, sobrecarregando uma única (ou poucas) máquinas).

  É recomendável projetar padrões de acesso a dados, de forma que o *Cluster* seja utilizado total e uniformemente.

  Para evitar pontos de acesso nas gravações, é importante projetar chaves de linha de forma que estas estejam na mesma região quando realmente precisam, mas, no quadro geral, os dados sejam gravados em várias regiões do *Cluster*.

  A Comunidade sugere algumas técnicas para evitar o *hotspotting*, que podem ser vistasaqui.

- **Seleção de número de Regiões:** Na criação de tabelas, no HBase, é possível definir explicitamente o número de regiões para a tabela ou o HBase calculará baseado em um algoritmo. É uma boa prática definir o número de regiões para a tabela e isto auxilia, inclusive, na melhoria de performance .

- **Escolha o número de *column family*:** Quase sempre, temos uma *row key* e uma *column family* mas algumas vezes podemos ter mais de uma *column family*. É boa prática manter o menor número possível de *column family* e nunca ultrapassar o número de 10 .

- ***Balanced Cluster*:** Um *Cluster* HBase é considerado "balanceado" quando a maioria dos *Region Server* tem um número igual de regiões. Você pode desejar ativar o *balancer*, que balanceará os *Clusters* a cada 5 minutos.

© Copyright Tecnisys 2026. All rights reserved.

- **Evitar execução de outro job no *Cluster* HBase:** Os *frameworks* de integração utilizados para configurar o HBase vêm com outras ferramentas como Hive, Spark, etc. Entratanto, o HBase consome, de forma intensiva, CPU e memória, e, esporadicamente realiza grande acesso sequencial I/O. Portanto, executar outros jobs irá resultar em um significativo decremento na performance.

- **Evitar compactação "maior", ou completa**

  HBase possui duas formas de compactação: a "menor", que combina um número configurável de *Hfiles* pequenos em um maior e a compactação "maior", quando todos os arquivos do *store* são lidos para uma região e gravados em um único arquivo. Isto consome tempo e impede que haja gravações na região até que finalize. É uma boa prática evitar a compactação "maior".

- **Regras práticas para o esquema de tabelas e dimensionamento de Region Servers:** Existem diferentes data sets com diferentes padrões de acesso e expectativas de níveis de serviço. Entretanto, a Comunidade apresenta algumas regras gerais que podem ser vistas aqui.

  Além dos itens acima, a Comunidade HBase disponibiliza uma série de recomendações para ajustes do desempenho do HBase, que podem ser lidas aqui.

Detalhes do Projeto Apache HBase

O HBase foi modelado a partir do Google BigTable e escrito em Java.

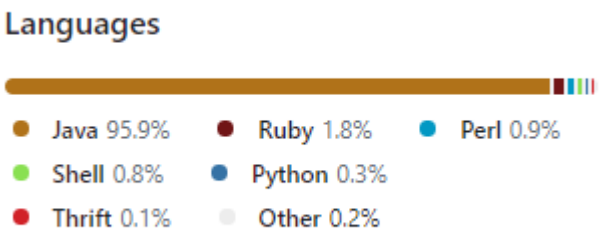


Figura 4 - Linguagens do HBase

Fontes

- HBase.apache.org
- HBase.apache.org/book.html

© Copyright Tecnisys 2026. All rights reserved.

- GitHub HBase
- Design de Esquemas - indicação comunidade
- Accordion:HBase respira com compactação na memória.
- Práticas e recomendações de criação de esquema

© Copyright Tecnisys 2026. All rights reserved.

Apache Iceberg



Table Formats

Table Formats são estruturas de tabelas utilizadas para organizar e gerenciar dados complexos, permitindo que sejam visualizados e interagidos como uma única "tabela" compreensível e acessível para vários usuários e ferramentas simultaneamente.

Os *Table Formats* adicionam uma camada de abstração semelhante a uma tabela sobre formatos de arquivo nativos, servindo como uma camada de metadados e fornecendo as primitivas necessárias para que os mecanismos de computação interajam com os dados armazenados de forma mais eficiente, garantindo a conformidade aprimorada com ACID, a capacidade de registrar dados transacionais com eficiência, a escalabilidade e a capacidade de atualizar ou excluir registros.

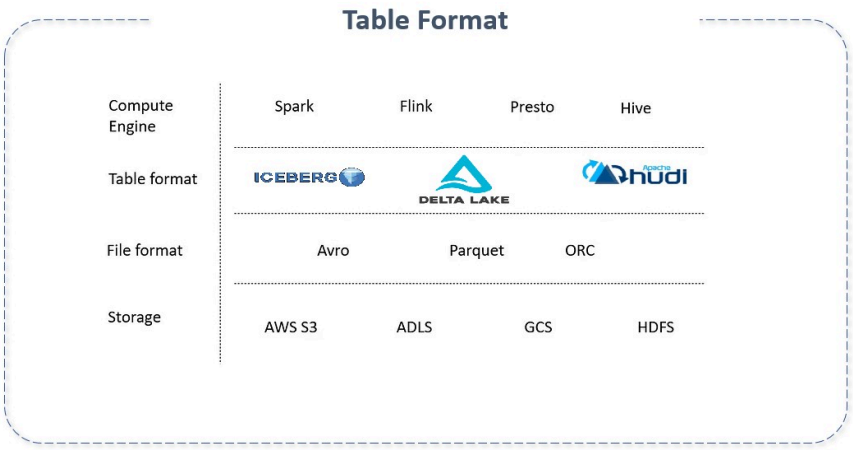


Table Formats adicionam uma camada de abstração aos arquivos de dados

Os *table formats* evoluíram como resposta a uma necessidade crítica: combinar as vantagens de gerenciamento de dados de "armazéns" (bancos de dados OLAP) com

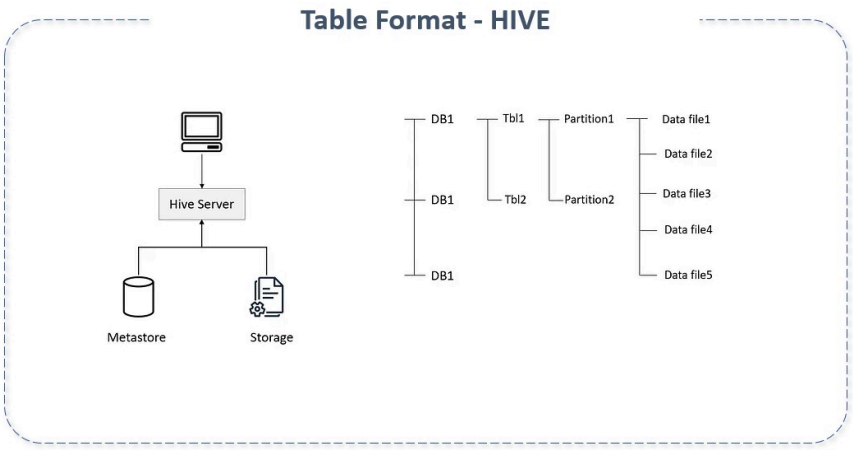
© Copyright Tecnisys 2026. All rights reserved.

maior escalabilidade e economia características dos *data lakes*. Em suma, combinam a versatilidade dos *data lakes* para lidar com dados brutos e semiestruturados com a capacidade de processar cargas de trabalho transacionais.

Em poucas palavras, *table formats* são uma forma de organizar arquivos de dados. Tentam trazer características de bancos de dados para o *data lake*. Sua principal meta é prover a abstração das tabelas para pessoas e ferramentas permitindo que interajam de forma eficiente com seus dados subjacentes.

Embora não fosse o nome usado à época, os *table formats* existem desde que System R, Multics e Oracle implementaram pela primeira vez o modelo relacional de Edgar Codd. Quando falamos que grupo de arquivos é uma coleção, estamos falando em *table formats*. Quando um arquivo é visto em formato de tabela, o diretório é visto como uma tabela, o que facilita seu rastreamento e o de seus arquivos. Isto é feito com a ajuda de sistemas de catálogo. Sob ele, arquivos são escritos em versões e aqueles mais recentes, correspondentes a uma tabela, são rastreados com a ajuda do metastore. O metastore é acessado para saber o estado atual da tabela e quais os arquivos relacionados.

O Apache Hive é um dos *table formats* mais antigos. Entretanto, como Hive foi *escrito* na era *pré-cloud*, não previu o armazenamento orientado a objetos, o que resultou em impactos no seu desempenho, uma vez que as tabelas de metadados do Hive crescem rapidamente. Para solucionar estas questões, novos formatos de tabelas foram criados.



Hive Table Format

© Copyright Tecnisys 2026. All rights reserved.

Apache Iceberg

O Apache Iceberg é um *table format* de código aberto para grandes conjuntos de dados analíticos, com uma especificação que garante compatibilidade entre linguagens e implementações. Adiciona tabelas a mecanismos de computação como o Spark, Trino, Flink, Hive, etc., usando um formato de tabela de alto desempenho que funciona exatamente como uma tabela SQL.

O Apache Iceberg permite o acesso a dados históricos em tempo real de forma coesa, garantindo integridade e consistência de dados. Sua principal inovação reside na capacidade de suportar operações de leitura, escrita, atualização e exclusão de dados sem reescrita de conjuntos inteiros de dados.

O Apache Iceberg teve uma adoção muito rápida entre grandes empresas como Apple, Netflix, Amazon, etc. A sua natureza completamente aberta, comprometendo várias empresas e a comunidade, torna-o ideal para a arquitetura de *data lake*.

Características do Apache Iceberg

O Apache Iceberg foi desenvolvido em 2017 pela Netflix, doado à Apache Software Foundation em novembro de 2018. Tornou-se um projeto de alto nível em 2020. É usado por inúmeras companhias incluindo o Airbnb, Apple, Linkedin, Adobe, etc. Foi criado especificamente para resolver problemas e desafios relacionados com tabelas formatadas de arquivos tradicionais em *data lakes*, como a evolução de dados e *schemas* e gravações concorrentes, de forma consistente, em paralelo. Introduz novos recursos que permitem que vários aplicativos (como o Dremio) trabalhem juntos nos mesmos dados de maneira transacional, de forma consistente, e definam informações adicionais sobre o estado dos conjuntos de dados à medida em que evoluem e mudam ao longo do tempo.

O Apache Iceberg adiciona tabelas para engines de computação como o Spark, Trino, Flink e Hive usando um formato de tabela de alta performance que funciona como uma tabela SQL.

Dentre suas principais características citamos:

- **Evolução de esquema:** Suporta a adição, exclusão, atualização ou renomeação, sem efeitos colaterais;

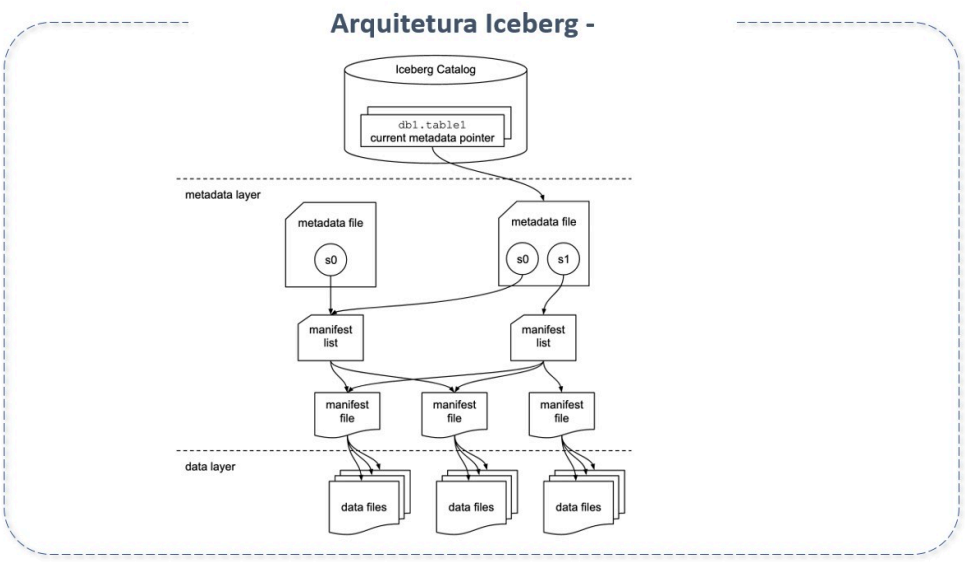
© Copyright Tecnisys 2026. All rights reserved.

- **Particionamento oculto:** Impede erros de usuário como causa de resultados incorretos de forma silenciosa ou consultas extremamente lentas;

- **Evolução do layout da partição**: Permite a atualização do layout de uma tabela à medida em que o volume de dados ou padrões de consulta mudam.

Arquitetura do Apache Iceberg

Na prática, o Apache Iceberg é uma especificação de formato de tabelas e um conjunto de APIs e bibliotecas que viabilizam a interação de mecanismos com tabelas, seguindo esta especificação.



Arquitetura de uma tabela Iceberg

Seus principais componentes são:

- **Catalogo Iceberg:** O catálogo é um repositório central onde é armazenada a referência ao arquivo de metadados de cada tabela. Seu objetivo principal é suportar operações atômicas para atualizar ponteiros. Para isto, guarda a localização atual do ponteiro de metadados (dentro do catálogo há uma referência ou ponteiro para o arquivo de metadados atual de cada tabela. O valor do ponteiro é o local do arquivo de metadados).

- **Camada de Metadados:** Os metadados são rastreados em três arquivos, os arquivos de metadados, a lista de manifestos e os arquivos de manifesto.

© Copyright Tecnisys 2026. All rights reserved.



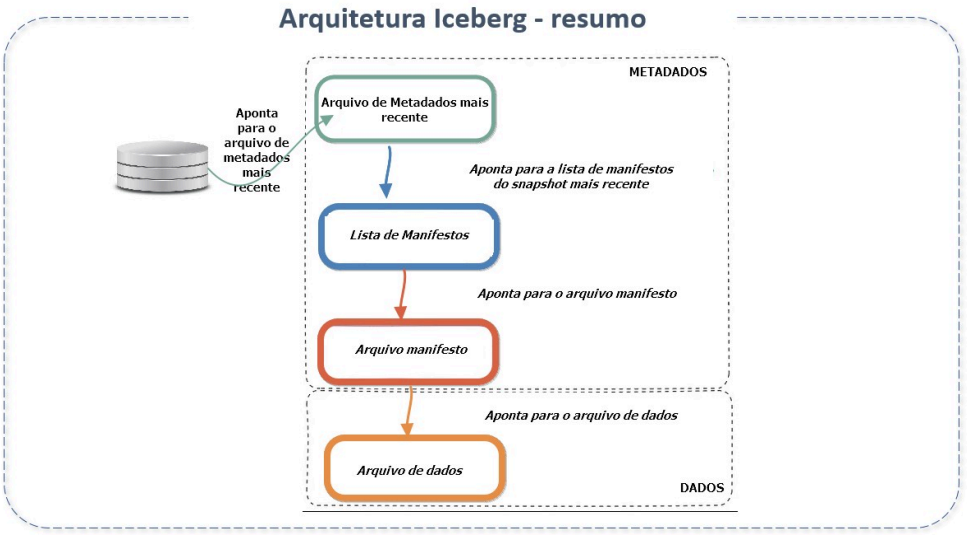
- Arquivos de metadados (*metadata file*): arquivo .json que contém informações sobre os metadados da tabela em determinado instante de tempo. Armazena o estado das tabelas. Nele encontramos detalhes sobre o *schema* da tabela, informações sobre partição, *id* do *snapshot* atual (current-snapshot_id), caminho para a lista de manifesto (manifest list), etc. Quando há mudança no estado das tabelas, cria um novo arquivo de metadados e substitui o antigo com um *swap* atômico. Possui as seguintes subseções:

    - *Snapshot*: É uma lista completa de todos os arquivos na tabela do *snapshot*. Representa o estado de uma tabela em determinado momento e é usado para acessar o conjunto completo de arquivos de dados na tabela. Contém informações sobre o *schema* da tabela, especificações de partição e localização da lista de manifesto;
    - *Schemas*: Todos os *schemas* de tabelas alterados são rastreados pela "matriz de *schemas*";
    - Especificações de partição: rastreia informações sobre a partição;
    - Ordens de classificação.
  - Lista de Manifestos (*manifest list*): Arquivo do tipo *avro* que contem todos os manifestos e suas métricas. Armazenam os metadados sobre os *manifestos* que compõem um *snapshot*, incluindo estatísticas de partição e contagens de arquivos de dados. Estas estatísticas são usadas para evitar a leitura de manifestos desnecessários à operação. Atua como um *link* entre o manifesto e o *snapshot*.

  - Arquivos de manifesto (*manifest file*): Mantém o controle sobre todos os arquivos de dados, juntamente com detalhes e estatísticas como seus formatos, localização e métricas.
- **Camada de Dados**

  - Arquivo de dados: O arquivo físico real. Onde de fato estão os dados.

© Copyright Tecnisys 2026. All rights reserved.



Resumo do Iceberg

O Apache Iceberg foi construído para tabelas enormes que podem ser lidas sem um mecanismo SQL distribuído. Cada tabela pode conter dezenas de petabytes de dados.

Recursos do Apache Iceberg

- **Recursos relacionados à experiência do Usuário**:

  O Apache Iceberg melhora a experiência do usuário no sentido em que:

  - Com o suporte à evolução do *schema* suporta adições, eliminações, atualizações, renomeações e reordenamento de colunas em uma tabela sem a necessidade de regravação da tabela. Isto torna a evolução do *schema* livre de efeitos indesejados, designando um ID único a cada coluna recém criada e automaticamente adicionando-a ao metadados da coluna. Também garante a exclusividade de cada coluna;

  - Com o particionamento oculto, garante que usuários não precisem mais saber o *layout* estrutural de arquivos antes de executar consultas. Isto evita erros do usuário que causam resultados silenciosamente incorretos ou consultas extremamente lentas. O Iceberg evolui o esquema e a partição da tabela à medida em que os dados são dimensionados;

  - Com o time travel, o recurso de controle de versão, o Iceberg garante o salvamento de qualquer alteração nos dados para referências futuras, seja adicionando, excluindo ou atualizando dados. Assim, qualquer problema com a

© Copyright Tecnisys 2026. All rights reserved.

versão de dados pode ser facilmente revertido para uma versão mais antiga e estável, garantindo que dados não sejam perdidos e que possam ser comparados ao longo do tempo.

- **Recursos de confiabilidade**:
    - O isolamento do *snapshot* garante que qualquer leitura do conjunto de dados veja um "instantâneo" consistente. Em essência, lê o último valor confirmado presente no momento da leitura. Isto evita conflitos. Após os "commits" o registro é atualizado e um novo *snapshot* é criado, refletindo a atualização mais recente;
    - Com os *Commits* atômicos, garante que os dados permaneçam consistentes em todas as consultas. Para evitar alterações parciais, qualquer atualização deve ser concluída no conjunto de dados, ou não salvará nenhuma alteração. Isto garante o retorno apenas de dados corretos, evitando que usuários visualizem dados incompletos ou inconsistentes;
    - As leituras são confiáveis, pois cada transação (atualização, adição, eliminação) cria um novo *snapshot*. Assim, os leitores podem utilizar as diversas versões mais recentes de cada atualização para criar uma consulta confiável para a tabela;
    - As operações são em nível de arquivo, diferentemente dos catálogos tradicionais, que rastreiam registros por posição ou nome, o que exige leitura de diretórios e partições antes da atualização de um único registro. No Apache Iceberg é possível direcionar diretamente um único registro e qualquer atualização do registro sem nenhuma alteração na pasta. Isto porque os registros estão armazenados em seus metadados.
- **Recursos de desempenho**:
    - Cada arquivo que pertence a uma tabela possui metadados armazenados para qualquer transação ocorrida, junto com estatísticas extras. Isto permite que usuários localizem apenas os arquivos que correspondem à consulta que desejam no momento;
    - O Iceberg utiliza dois níveis de metadados para rastrear os arquivos em um *snapshot* - os arquivos de manifesto e a lista de manifestos. O primeiro nível contém os arquivos de dados, junto com seus dados de partição e estatísticas

© Copyright Tecnisys 2026. All rights reserved.

em nível de coluna. O segundo armazena a lista de *snapshots* do manifesto com um intervalo de valores para cada partição;
    - Para obter um rápido "scan planning", o Iceberg usa valores mínimo e máximo da partição na lista de manifestos para filtrar os manifestos. Posteriormente, ele lê todos os manifestos retornados para obter o arquivo de dados. Assim, é possível planejar sem ler todos os arquivos de manifesto, usando a lista de manifestos para restringir o número de arquivos de manifestos necessários à leitura.

Com isto, é possível alcançar consultas eficientes e econômicas em arquivos.

Boas Práticas no Uso do Apache Iceberg

O Apache Iceberg é uma ferramenta poderosa para gerenciamento de grandes conjuntos de dados analíticos. Para maximizar sua eficiência e eficácia, aqui estão algumas boas práticas:

- **Estrutura de Dados e Esquemas**: Mantenha os esquemas de dados simples e evolutivos. Utilize a funcionalidade de evolução de esquema do Iceberg para manter a compatibilidade e minimizar as interrupções;

- **Particionamento Eficiente**: Utilize particionamento lógico para organizar os dados de maneira que otimize as consultas mais comuns. O particionamento oculto do Iceberg pode ajudar a evitar problemas de desempenho;

- **Gerenciamento de Metadados**: Mantenha os metadados limpos e atualizados. Isso inclui remover *snapshots* antigos e arquivos de manifesto não utilizados para evitar sobrecarregar o catálogo;

- **Testes e Validação**: Implemente testes rigorosos ao evoluir esquemas ou modificar tabelas para garantir a integridade dos dados;

- **Monitoramento e Otimização**: Monitore regularmente o desempenho das consultas e otimize as tabelas conforme necessário. Isso pode incluir ajustar o particionamento ou modificar índices;

- **Documentação**: Mantenha uma documentação clara sobre as estruturas de tabelas, esquemas e qualquer lógica de negócios associada para facilitar a manutenção e a colaboração;

© Copyright Tecnisys 2026. All rights reserved.



- **Segurança e Controle de Acesso**: Implemente controles de acesso adequados e práticas de segurança para proteger os dados;
- **Uso de Recursos e Custo**: Esteja ciente do uso de recursos e custos associados, especialmente em ambientes de nuvem. Otimize o uso de armazenamento e computação para manter a eficiência de custos;
- **Atualizações e Compatibilidade**: Mantenha-se atualizado com as versões mais recentes do Apache Iceberg para aproveitar melhorias e correções de segurança;

Utilize o Apache Iceberg de acordo com essas práticas pode melhorar significativamente a gestão de dados e a eficiência operacional.

Quando usar o Apache Iceberg

- **Grandes Conjuntos de Dados**: Quando estiver lidando com petabytes de dados em tabelas enormes;
- ***Data Lakes* Modernos**: Para arquiteturas modernas de data lake que necessitam de gerenciamento robusto de dados e esquemas;
- **Escalabilidade e Confiabilidade**: Quando a escalabilidade e a confiabilidade são críticas para a operação e análise de dados;
- **Transações ACID em *Data Lakes***: Para suportar transações ACID em *data lakes*, garantindo a integridade dos dados;
- **Evolução de Esquema**: Quando há necessidade de evoluir o esquema de dados sem interrupção ou perda de dados;
- **Atualizações e Deleções**: Se precisar fazer atualizações e deleções eficientes em grandes conjuntos de dados;
- **Trabalho com Múltiplos Formatos de Dados**: Se trabalha com vários formatos de dados e precisa de uma camada de abstração uniforme;
- **Leituras e Escritas Concorrentes**: Quando precisar de suporte para leituras e escritas concorrentes sem bloqueio;

© Copyright Tecnisys 2026. All rights reserved.



- **Compliance e Governança de Dados**: Para atender a requisitos rigorosos de *compliance* e governança de dados;
- **Integração com Plataformas de Análise**: Se deseja integrar com plataformas de análise como Spark, Trino, Flink e outras;
- **Snapshot e Traveling de Dados**: Quando a funcionalidade de snapshot e traveling de dados é necessária para auditoria ou *rollback*;
- **Melhoria na Performance de Leitura**: Para melhorar a performance de leitura através de indexação e otimizações de arquivo;

Estes itens destacam cenários ideais para a implementação do Apache Iceberg, maximizando seus recursos para gerenciamento eficiente de *data lakes*.

Quando não usar o Apache Iceberg

O Apache Iceberg não é a ferramenta adequada para:

- **Pequenos volumes de dados**: Se o universo de dados for pequeno, que não exija o uso de *data lake*, o Apache Iceberg não trará benefícios.
- **Ingestão em tempo real**: Apache Iceberg não oferece suporte a ingestão de dados em tempo real porque usa o processamento em lote.
- **Estrutura centralizada**: Se o objetivo não é usar uma estrutura de computação distribuída, o Apache Iceberg não é a escolha ideal, pois foi projetado para usar uma estrutura de computação distribuída para processar dados.

Detalhes sobre o projeto

O Apache Iceberg é desenvolvido principalmente em Java. Portanto, uma boa compreensão de Java é essencial para contribuir efetivamente para o projeto ou para personalizá-lo. Isso também implica a necessidade de manter as práticas padrão de codificação Java, como o gerenciamento eficiente de exceções e a utilização de bibliotecas Java comuns para operações de I/O e manipulação de dados.

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**

© Copyright Tecnisys 2026. All rights reserved.



> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **1.10.0**, distribuída pelo Helm Chart `tdp-iceberg` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fonte(s):

- Iceberg.apache.org

© Copyright Tecnisys 2026. All rights reserved.

JupyterLab

Recurso disponível somente a partir da versão 3.0.

Interface Web de próxima geração

JupyterLab é uma aplicação avançada, extensível e rica em recursos para criação e edição de notebooks computacionais, que integra código, explicações em linguagem natural, dados e visualizações.

Ele faz parte do Project Jupyter, iniciativa que provê padrões e ferramentas para computação interativa e ciência reprodutível.

Embora seja acessado via navegador, não se trata apenas de uma “interface web”, mas de um ambiente completo de trabalho, com múltiplos painéis, extensões, suporte a diferentes linguagens e integração com ecossistemas de dados e ciência.

Lançado oficialmente em 2018, o JupyterLab foi financiado por fundações como a Gordon and Betty Moore Foundation e The Alfred P. Sloan Foundation, com o objetivo de expandir as funcionalidades das ferramentas Jupyter e criar uma experiência de desenvolvimento interativa mais completa.

Embora seja "irmão" de outros aplicativos de criação de notebooks sob o mesmo nome do Projeto Jupyter, como o Jupyter Notebook e Jupyter Desktop, oferece uma experiência mais avançada, mais rica e personalizável, diferenciando-se das outras ferramentas pela sua extensibilidae, modularidade e adoção ampla na comunidade científica e educacional.

O que ele faz

- **Autoria de notebooks**: combina células de código executável, texto explicativo (Markdown/LaTeX), visualizações e resultados interativos.
- **Ambiente unificado**: permite abrir notebooks, terminais, arquivos de texto, JSON, CSV e Markdown na mesma janela.

© Copyright Tecnisys 2026. All rights reserved.

- **Execução interativa**: conecta-se a kernels de múltiplas linguagens (Python, R, Julia, Scala etc.).
- **Exploração de dados**: suporta bibliotecas de visualização avançadas (Matplotlib, Plotly, Bokeh, Vega, 3D).
- **Compartilhamento e reprodutibilidade**: notebooks são documentos portáveis e versionáveis.
- **Extensibilidade**: adiciona dashboards, renderizadores de formatos especiais, integração com Git, conectores de dados etc.

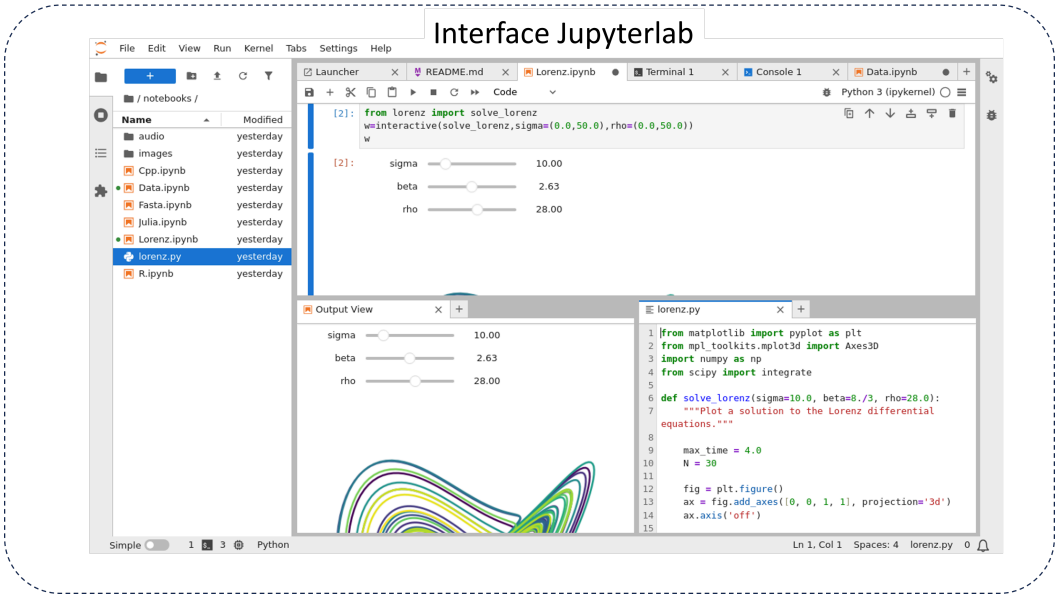


Figura 1 - Interface do JupyterLab

Características do JupyterLab:

- **Interface modular**: múltiplas abas e painéis arrastáveis, layouts customizáveis.
- **Paleta de comandos**: central de atalhos, semelhante a IDEs modernas.
- **Gestão de kernels e sessões**: permite visualizar, reiniciar ou encerrar kernels ativos.
- **Gerenciador de arquivos**: navegação e manipulação de diretórios no servidor.
- **Terminais embutidos**: acesso a shell integrado.

© Copyright Tecnisys 2026. All rights reserved.



- **Integração com extensões**: desde suporte a novos formatos de dados até painéis customizados.
- **Compatibilidade retroativa**: notebooks antigos do Jupyter continuam funcionais.

Arquitetura do JupyterLab

O JupyterLab se apoia em uma arquitetura em camadas.

- **Frontend (JupyterLab UI)**: Construído em TypeScript, baseado em widgets modulares, roda no navegador e gerencia a interação com o usuário.
- **Jupyter Server** Backend responsável por:
  - Gerenciar arquivos e diretórios.
  - Servir a aplicação web.
  - Intermediar comunicação via WebSockets/HTTP entre UI e kernels.
- **Kernels**: Processos independentes para execução de código em diferentes linguagens. O mais popular é o IPython Kernel, mas há dezenas disponíveis (R, Julia, Scala, Java, Rust).
- **Protocolos de Mensagens**: Comunicação entre UI e kernels segue o Jupyter Messaging Protocol, baseado em JSON e ZeroMQ, padronizando requisições, respostas e eventos.

Seus principais componentes são:

- **Notebook**: documento interativo com células de código e markdown.
- **Console de código**: REPL avançado conectado a um kernel, útil para testes rápidos.
- **Editor de texto**: com syntax highlighting, útil para scripts, JSON, configs.
- **Terminal**: shell Unix dentro do navegador.
- **File Browser**: gerencia arquivos, diretórios e arrastar/soltar.
- **Launcher**: tela inicial para abrir novos notebooks, consoles ou arquivos.

© Copyright Tecnisys 2026. All rights reserved.

- **Extensões**: módulos de terceiros ou oficiais que expandem a interface e funcionalidades.

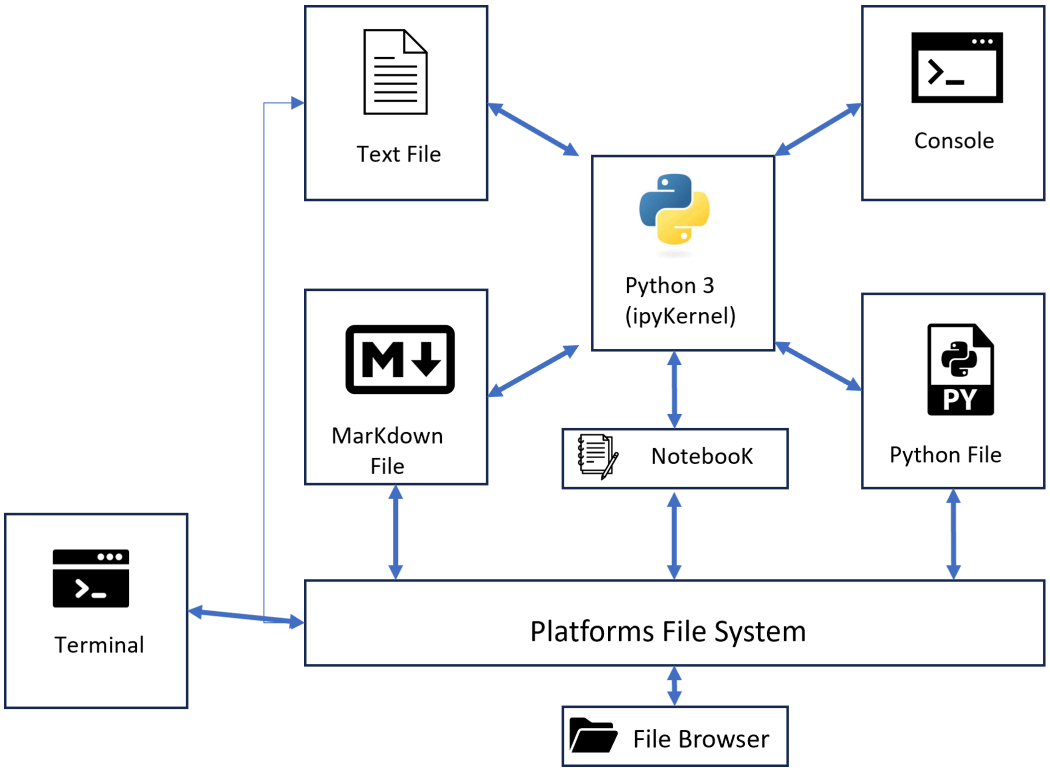


Figura 2 - Arquitetura do JupyterLab

Recursos do JupyterLab

- **Layouts flexíveis**: organizar notebooks, consoles e arquivos lado a lado.
- **Visualizações avançadas**: gráficos interativos, modelos 3D, dashboards.
- **Suporte multilíngue**: kernels para dezenas de linguagens.
- **Colaboração em tempo real**: com JupyterLab RTC (experimental).
- **Exportação**: conversão de notebooks em PDF, HTML, slides (via nbconvert).
- **Integração com Git e CI/CD**: extensões permitem versionamento integrado.

Boas Práticas no Uso do JupyterLab

- **Gerenciar ambientes**: mantenha as dependências de cada projeto em um ambiente isolado e reprodutível, utilizando ferramentas adequadas ao ecossistema (como gerenciadores de ambientes, empacotadores ou contêineres).

© Copyright Tecnisys 2026. All rights reserved.



- **Documentar notebooks**: combine código com explicações claras, títulos e descrições.
- **Evitar saídas pesadas**: limpe outputs antes de versionar para reduzir repositórios.
- **Controlar versões**: integre com Git; notebooks podem ser armazenados no GitHub com renderização nativa.
- **Segurança**: não execute notebooks de fontes desconhecidas sem revisão.
- **Estrutura modular**: use notebooks como protótipos, e quando estáveis, mova funções para módulos Python.
- **Exportar e reportar**: utilize ferramentas de conversão de formatos para publicar notebooks em PDF, slides ou HTML(por exemplo, nbconvert, Quarto, Jupyter Book, entre outras).
- **Extensões com cautela**: instale apenas o necessário e mantenha atualizado.

Quando usar o JupyterLab

- **Prototipagem rápida de código e ideias**

  Quando você precisa testar trechos de código de forma interativa, com feedback imediato.

  Ideal para data exploration, análise ad hoc, e ensaios de algoritmos.

  Diferencial: você mantém código + explicação + resultado no mesmo documento.

- **Análise e exploração de dados**

  Excelente para ETL leve, gráficos exploratórios e estatísticas rápidas.

  Integra bem com bibliotecas como Pandas, NumPy, Matplotlib, Plotly, Seaborn, Bokeh.

  Recursos como gráficos interativos, widgets e notebooks parametrizados tornam a exploração mais rica do que em um script estático.

- **Educação, treinamento e documentação técnica**

© Copyright Tecnisys 2026. All rights reserved.



Facilita aulas, tutoriais e cursos: o estudante lê a teoria, executa o código e vê o resultado no mesmo fluxo.

Muito usado em MOOCs, bootcamps e livros digitais de ciência de dados.

Permite distribuir materiais reprodutíveis (aluno executa e obtém o mesmo resultado).

- **Ciência reprodutível e pesquisa acadêmica**

  Ideal quando é essencial documentar os passos analíticos (código + dados + explicações).

  Suporta exportar para artigos (PDF, LaTeX, HTML).

  Permite reproduzir experimentos, compartilhar pipelines e embutir visualizações.

- **Colaboração em equipes de dados**

  Ambientes de dados compartilhados (on-premises ou cloud) podem expor JupyterLab como hub de notebooks.

  Com JupyterHub ou extensões de colaboração, várias pessoas acessam o mesmo servidor.

  Evita problemas de “funciona na minha máquina” (ambientes centralizados).

- **Projetos que envolvem múltiplos arquivos e linguagens**

  Diferente do Jupyter Notebook clássico, o JupyterLab abre vários arquivos lado a lado (notebook + CSV + Markdown + terminal).

  Útil em workflows mais complexos, onde você precisa editar scripts auxiliares, JSON de configuração ou mesmo documentos Markdown, sem sair do ambiente.

- **Quando é necessário extensibilidade**

  Para personalizar a interface ou integrar ferramentas externas (Git, MLflow, dashboards, renderizadores de formatos específicos).

© Copyright Tecnisys 2026. All rights reserved.



O ecossistema de extensões permite que o JupyterLab se torne quase uma IDE leve e focada em notebooks.

Quando não usar JupyterLab (limitações práticas)

- **Desenvolvimento de software grande/complexo**: para projetos robustos, com muitos módulos e testes, IDEs como PyCharm ou VS Code oferecem melhor suporte.
- **Execuções muito longas em notebooks**: se o código roda horas/dias, é melhor colocar em scripts versionados ou pipelines (Airflow, Prefect, Luigi).
- **Produção/Deploy**: JupyterLab é ambiente de prototipagem e análise, não de execução contínua em produção.

Detalhes sobre o projeto

Em essencia, enquanto Python é a linguagem dominante para data science e computação dentro do JupyterLab, Typescript(com React e Lumino, antigo PhosphorJS) tem um papel crucial na construção da plataforma (front end) e extensão de seus recursos.

O Backend é desenvolvido em Python, servindo arquivos e interface web, gerenciando sessões e autenticação e intermediante a comunicação com os kernels via protocolo Jupyter.

Em relação aos kernels, o IPython kernel é o mais usado, também escrito em Python, extensível a muitas linguagens(R, Julia, Scala, Java, Rust, etc.).

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **5.3.0**, distribuída pelo Helm Chart `tdp-jupyter` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

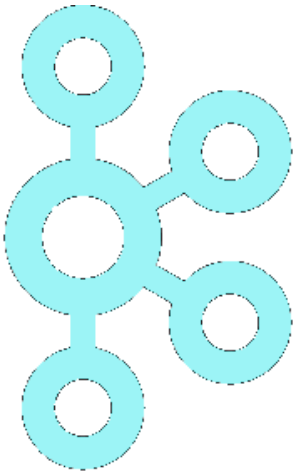
© Copyright Tecnisys 2026. All rights reserved.

Fonte(s):

- JupyterLab.readthedocs.io

© Copyright Tecnisys 2026. All rights reserved.

Apache Kafka



Streaming de Dados

Métodos legados de processamento em lote exigem que os dados sejam coletados em forma de lote antes de serem processados, armazenados ou analisados.

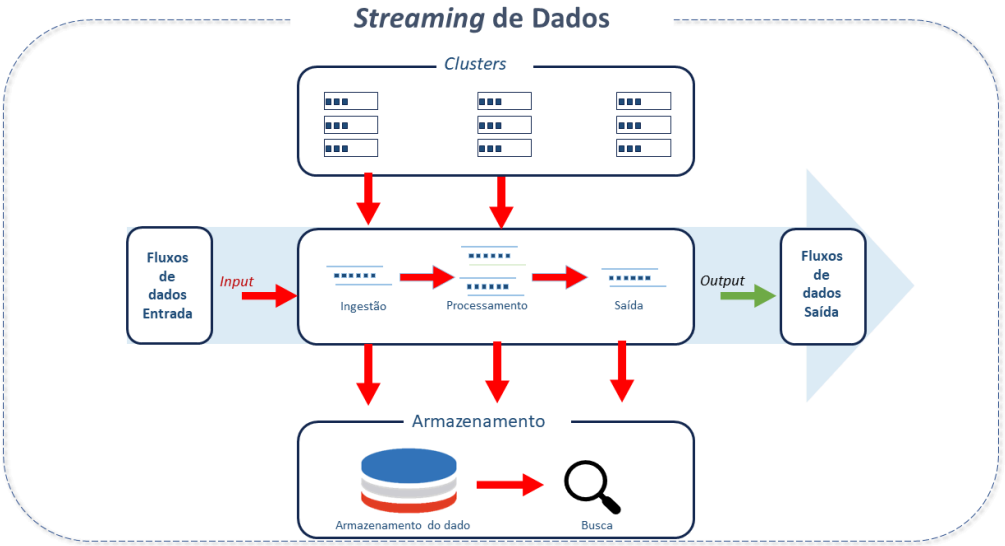
Por outro lado, dados de "streaming" fluem continuamente, e podem ser tratados imediatamente, permitindo decisões dinâmicas, contextuais e em tempo real, capacitando, assim, as empresas a explorar todo o valor e potencial de seus dados e aplicativos.

Streaming de Dados é a captura de fluxos de dados(elementos de dados ordenados no tempo) em tempo real a partir das mais diversas fontes de eventos como databases, sensores, mobiles, nuvem e aplicativos de software.

Os fluxos(*streams*) capturados podem ser tratados em tempo real ou retrospectivamente e roteados para diferentes tecnologias, conforme necessário. Podem ser armazenados para posterior recuperação, manipulação ou processamento.

O Streaming de dados envolve tecnologias que surgiram nos últimos anos. Um sistema de análise de dados em tempo real, por exemplo, está preparado para alta geração de dados, com base em eventos que rapidamente se espalham pela rede.

© Copyright Tecnisys 2026. All rights reserved.



Streaming de Dados

Características do Apache Kafka

O Kafka é a plataforma de *streaming* de eventos mais comumente utilizada. Kafka é usado para coletar, processar, armazenar e integrar dados em escala. Possui numerosos casos de uso, incluindo registro distribuído, processamento de fluxos, integração de dados e publicação/subscrição de mensagens.

Originalmente criada como solução para um software do Linkedin, que coletava dados de atividades do usuários no portal de dados e os usava para mostrar num portal web, a plataforma foi construída como um sistema distribuido, tolerante a falhas, combinando três funcionalidades:

- Publicação (*write*) e Subscrição (*read*) de fluxos de eventos, incluindo importações e exportações contínuas do dado a partir de outro sistema.
- Armazenamento de fluxos de eventos de forma duradoura e confiável pelo tempo necessário.
- Processamento de fluxos de eventos a medida que ocorrem ou de forma retrospectiva.

Kafka foi construido com as seguintes premissas:

- Baixo acoplamento entre producers e *consumers*.
- Persistência de dados para suportar uma variedade de cenarios de consumo e tratamento de falhas.
- Máxima taxa de transferência "end-to-end" com componentes de baixa latência.

© Copyright Tecnisys 2026. All rights reserved.



- Gerenciamento de diversos formatos e tipos de dados usando formatos de dados binários.

> **NOTE**
>
> Kafka é comumente usado em sua arquitetura de processamento *stream*. Com sua semântica *reliable message delivery* (entrega confiável de mensagens), auxilia no consumo de altas taxas de eventos. Provê, ainda, recursos de *replay* de mensagens para diferentes tipos de *consumers*.

Arquitetura do Apache Kafka

- **Sistema de Comunicação**: Kafka é um sistema distribuído que consiste em servidores e clientes se comunicando por meio de protocolo de rede TCP de alta performance.
  Pode ser implementado em hardware *bare-metal*, máquinas virtuais e containers, assim como no ambiente em nuvem.
  O *Cluster Kafka* é altamente escalável e tolerante a falhas.
  Seu modelo de comunicação utiliza o sistema WAL (Write-ahead log) no qual toda mensagem é publicada em arquivos *log* antes de disponibilizada para aplicações *Consumer* para que *Subscribers* e *Consumers* possam ler no momento apropriado.
  Kafka simplifica a comunicação entre sistemas agindo como um *hub* centralizado de comunicação.
  O padrão de comunicação implementado é o *publish-subscribe*.
  Projetado como uma biblioteca de cliente simples e leve, pode ser facilmente incorporado a qualquer aplicativo Java e integrado a qualquer pacote.
  Não possui dependências externas na camada de mensagens.
  A camada de mensagens particiona os dados para armazená-los e transportá-los.
  Este particionamento é que permite a localidade, escalabilidade, alto desempenho e tolerância a falhas.

- **Os principais componentes do Kafka são:**

  - **Clientes**: Os clientes viabilizam a escrita de microsserviços e aplicações distribuidas que lêem, gravam e processam *streams* de eventos em paralelo, em escala e de uma forma tolerante a falhas, mesmo diante de problemas de máquina ou rede.
    Kafka traz alguns clientes incluídos, os quais são aumentados por vários outros

© Copyright Tecnisys 2026. All rights reserved.

entregues pela comunidade: Clientes estão disponíveis para Java e Scala, incluindo a biblioteca de alto nível *Kafka Streams*, para Go, Python, C/C++ e muitas outras linguagens e APIs REST.

- **Servidores:** Kafka é executado como um *Cluster* de um ou mais servidores que pode estender-se a múltiplos Datacenters ou *Cloud-Regions*.
  Alguns destes servidores, denominados *brokers*, formam a camada de armazenamento.

  - **Brokers:** Um *Cluster* Kafka tipico consiste de multiplos *brokers*. Isto ajuda no balanceamento de carga nas leituras e gravações no cluster.
    Cada broker é *steteless*(não permanece dedicado à conexão) e conta com o Zookeeper para manter seu estado.
- **Topics: (tópicos)** São a forma de organização dos eventos. Sua função é similar às tabelas de um Database (agrupam dados relacionados) sem, entretanto, impor um esquema particular.
  Armazenam as mensagens em dados *raw bytes* (bytes não codificados), o que os torna muito flexíveis para tratamento, suportando dados homôgeneos e heterogêneos.
  São categorizados em *partitions*.
  Fisicamente, cada tópico é distribuído por diferentes *brokers* , os quais hospedam uma ou duas *partitions* para cada tópico.

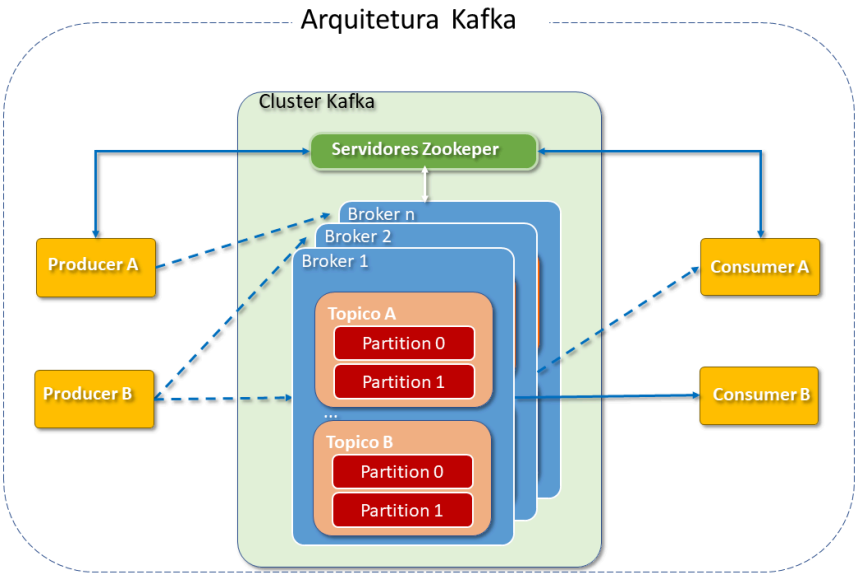
- **Partitions (partições)**: As partições armazenam mensagens na sequência em que chegam. Eventos com a mesma chave(id) são gravados na mesma partição.
  O número de partições de um tópico é configurável, assim como o seu tamanho. Contar com mais partições em um tópico geralmente traduz-se em mais paralelismo e *throughput* .
  Os *pipelines* Kafka devem ter um número uniforme de partições por *broker* e todos os *topicos* em cada máquina.
  Em cada partição, um dos *brokers* é o lider e zero ou mais são seguidores. Os lideres gerenciam as requisições de leitura ou gravação para suas respectivas partições.
  Os seguidores replicam o lider mas não interferem no seu trabalho, atuando, como um "backup".
  Um deles será escolhido para substituí-lo em caso da falha. Cada Cluster Kafka pode ser simultaneamente um lider de alguma partição de *topico*, ou seguidor em outras.
  Assim, a carga em qualquer servidor é balanceada igualmente.

© Copyright Tecnisys 2026. All rights reserved.

A eleição do lider é feita com ajuda do Zookeeper, que gerencia e coordena os *brokers* e *consumers* .
O Zookeeper acompanha qualquer adição ou falha de broker no *Cluster*, notificando seu estado aos *producers* ou *consumers* das filas Kafka.
Também auxilia *producers* e *consumers* na coordenação dos *brokers* ativos, registrando quais são os lideres para qual partição de tópico e passando esta informação para *producers* e *consumers*.

- ***Producers*** São as aplicações responsáveis por enviar dados para a partição do tópico para o qual está produzindo dados.
  O *Producer* não grava dados na partição. Apenas cria solicitações de gravação para mensagens e as envia para o *broker* líder.
  Dependendo da configuração, o *producer* aguarda por uma confirmação de mensagens.

- **Consumers:** São aplicações ou processos que subscrevem (lêem e processam) os eventos.
  Buscam mensagens a partir dos arquivos de *log* pertencentes a uma partição do tópico.
  São eles que distribuem o trabalho em múltiplos processos.



Arquitetura do Apache Kafka

Kafka-UI

O Kafka-UI é uma interface web versátil, rápida e leve para gerenciamento e monitoramento de *clusters* Kafka.

© Copyright Tecnisys 2026. All rights reserved.



Trata-se de uma ferramenta open source que auxilia na observação dos fluxos de dados, detecção e solução de problemas, gestão e análise de desempenho.
Seu *dashboard* permite o rastreamento das principais métricas de *Brokers*, Tópicos, Partições, e Produção e Consumo de Eventos.

Principais Recursos do Kafka-UI

- **Exploração de mensagens** - navegue pelas mensagens de tópicos usando com Avro, Protobuf, JSON ou codificação de texto simples;
- **Visualização de grupos de consumidores** — visualização, por partição, de offset estacionados (*parked offsets*) e atrasos
- **Assistente de configuração** — configuração de clusters Kafka através de uma interface web (UI)
- **Gerenciamento de vários clusters** — monitoramento e gerenciamento de todos os clusters em um só lugar
- **Monitoramento de desempenho com painel de métricas** - rastreamento e exibição das principais métricas do Kafka
- **Visualização de Kafka Brokers** - visualização de atribuições de tópicos e partições, e status do controlador
- **Visualização de tópicos do Kafka** - visualização de contadores, status da replicação e configurações personalizadas
- **Configuração dinâmica de tópicos** — criação e configuração de novos tópicos com configuração dinâmica
- **Controle de acesso baseado em função** - gerenciamento de permissões para acessar a UI com precisão granular
- **Mascaramento de dados** - ofuscamento de dados confidenciais de mensagens de tópicos

Recursos do Apache Kafka

- **Ferramentas de código aberto para grandes ecossistemas**:

© Copyright Tecnisys 2026. All rights reserved.



Vasta gama de ferramentas orientadas pela comunidade.

- **APIs Kafka:** Adicionalmente às ferramentas de linha de comando, Apache Kafka disponibiliza cinco APIS para Java e Scala nas tarefas de gerenciamento e administração:
    - **API Admin** para gerenciar e inspecionar topicos, *brokers* etc.
    - **Producer-API** Para publicação(gravação) de fluxos de eventos em um ou mais topicos.
    - **Consumer-API**: Para subscrição(leitura) de um ou mais topicos e processamento do fluxo de eventos produzido.
    - **API Kafka Streams**: Uma vez o dado armazenado como evento no Kafka, pode ser processado com a biblioteca cliente *Kafka Streams* para Java/Scala, que permite a implementação de aplicações e microsserviços *real- time* de missão critica, onde a(s) entrada(s) e/ou saída(s) são armazenadas nos *tópicos* Kafka. O *Kafka Streams* combina a simplicidade de escrever e implantar aplicativos Java e Scala no lado cliente com os benefícios da tecnologia do *Cluster* no lado do servidor, tornando esses aplicativos altamente escaláveis, elásticos, tolerantes a falhas e distribuídos.
    - **Kafka Connect API**: Para construção e execução de conectores de importação/exportação de dados que consomem ou produzem fluxos de eventos.

- **Monitoramento Operacional**: Kafka é frequentemente usado para Monitoramento Operacional, que envolve estatísticas de aplicativos distribuídos para produção de fontes centralizadas de dados operacionais.

- **Solução de Agregação de *Logs*** Kafka pode ser usado como Solução de Agregação de *Logs*, coletando arquivos de *log* físicos de servidores e colocando-os em um local centralizado (como o HDFS) para processamento.
  Em comparação aos sistemas "centrados" em *log* como Flume e Scribe, oferece desempenho similar, com garantias mais fortes de durabilidade, devido à replicação e menor latência de ponta a ponta .

- ***Event Sourcing*** Kafka pode ser utilizada em *Event Sourcing* , armazenando alterações no estado do aplicativo como uma sequência de eventos que podem

© Copyright Tecnisys 2026. All rights reserved.

não apenas ser consultados mas também ter seu log usado para reconstruçao de estados passados e mudanças retroativas.

- **Compactação de *Log*** Kafka pode servir como um *commit-log* externo para sistemas distribuídos.
  O *log* ajuda a replicar dados entre os nós e atua como mecanismo de *ressincronização* para que os nós com falha restaurem seus dados.
  O recurso de Compactação de *Log* do Kafka suporta este uso. Neste sentido, o Kafka é similar ao projeto **Apache Bookeeper**.

- **Replicação** O Apache Kafka replica o *log* para cada partição de tópico por um número configurável de servidores (este fator de replicação pode ser definido em bases topyc-por-topico).
  Isto auxilia no *failover* automático para estas replicas quando o servidor do *Cluster* falha de modo que as mensagens permanecem disponiveis na presença de falhas.

- **Quotas** O *Cluster* Kafka tem a capacidade de "impor" cotas em solicitações para controlar os recursos do agente usados pelo cliente.
  Dois tipos de cota de cliente podem ser aplicados para cada grupo de clientes que compartilha uma cota:

  - As cotas de largura de banda da rede : que definem os limites da taxa de bytes (a partir de 0,9).
  - As cotas de taxa de solicitação: que definem os limites de utilização da CPU como um percentual da rede e dos *threads* de E/S (a partir de 0,11).

Boas Práticas para Apache Kafka

- **Validação de dados:** Durante a gravação de um sistema *producer*, é essencial a realização de testes de validação dos dados que serão gravados no *Cluster* (valores não nulos para campos chave por exemplo).

- **Exceções:** Durante a gravação de um *Producer* ou *Consumer* é importante que sejam definidas classes de exceção e as ações a serem tomadas em conformidade com os requisitos dos negócios.
  Isto ajuda não somente no *debug*, mas mitiga riscos. (alertas para situações definidas, por exemplo).

- **Numero de *retries*(novas tentativas):** Em geral existem dois tipos de erro na aplicação *producer*: erros "solucionáveis" com nova tentativa (como network

© Copyright Tecnisys 2026. All rights reserved.

timeouts e "lider não disponivel") e erros que precisam ser tratados pelo *producer*. Configure o número de *retries* ajuda a mitigar riscos relacionados à perda de mensagens devido a erros do *Cluster* kafka ou rede.

- **Número de *bootstrap* URLS:** É importante ter mais que um *broker* listado no *bootstrap broker configuration*, no programa *producer*. Isto auxilia *producers* a ajustar-se quando houver falhas por causa de indisponibilidade de um broker.
  Os *producers* tentam usar todos os *brokers* listados até encontrar um com o qual possa se conectar.
  O ideal é listar todos os brokers no *Cluster* kafka para acomodar todas as falhas de conexão. Entretanto, em caso de *Clusters* muito grandes, pode-se escolher um menor número que possa representar significativamente os *brokers* do *Cluster*.
  Atente que o número de *retries* pode afetar a latência "end-to-end" e causar duplicação de mensagens na fila Kafka.

- **Novas partições em topicos existentes:** Novas partições em topicos existentes devem ser evitadas quando usando particionamento baseado em chaves para distribuição das mensagens.
  Adicione novas partições pode mudar o *hash* calculado para cada chave pois considera o número de partições como uma de suas entradas.
  Acabariam existindo partições diferentes para uma mesma chave.

- **Rebalanceamentos:** Sempre que um novo consumidor ingressa em grupos de *consumers* ou um antigo fica inativo, um reequilíbrio de *partições* no *Cluster* é acionado.
  Sempre que um *consumer* estiver perdendo a propriedade de sua partição, é importante o *Commit* dos *offsets* do último evento que recebeu do Kafka.

- **Commit offsets na hora certa:** No caso de *commit offset for messages* é necessário fazê-lo no momento correto. Um aplicativo em processamento *batch* toma mais tempo para completar o processamento.
  Não é uma regra, mas se o processamento durar mais que um minuto, é razoável realizar o *commit the offset* em intervalos regulares para evitar processamento duplicado de dados no caso de falha da aplicação.
  Para aplicações mais criticas onde esta duplicação possa causar problemas financeiros, o tempo de *commit offset* deve ser o menor possível se *throughput* não for um fator importante.

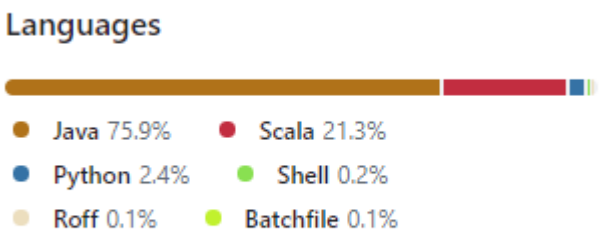
Outras recomendações

© Copyright Tecnisys 2026. All rights reserved.

O **blog.kafka.br** disponibiliza uma série de discussões e recomendações que valem a pena ser conhecidas.

Detalhes do Projeto Apache Kafka

O Kafka foi escrito nas linguagens de programação Java e Scala.



Linguagens do Kafka

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **4.1.0**, distribuída pelo Helm Chart `tdp-kafka` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fonte(s):

- kafka.apache.org
- conceitos centrais do kafka - kafka.apache.org
- blog.kafkabr
- cwiki.apache.org/confluence/display/kafka
- github.com/apache/kafka

© Copyright Tecnisys 2026. All rights reserved.

Kerberos

Autenticação e Propagação de Identidade

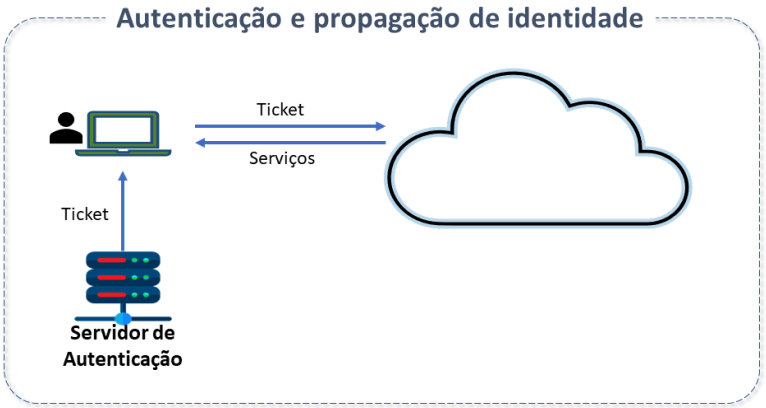


A autenticação pode ser categorizada em dois tipos:

- **Autenticação de Serviços**: Verifica a identidade entre diferentes componentes de serviço como HDFS, YARN, MapReduce, etc.

- **Autenticação de Usuário**: Um processo que permite a um dispositivo verificar a identidade de um usuário/cliente que se conecta a um recurso de rede. Sem a autenticação de usuário, o serviço simplesmente confia na informação de identidade fornecida pelo cliente.

  Na maioria dos cenários, uma *senha* é utilizada como prova de identidade. No entanto, é necessário prevenir a interceptação ou "escuta" desta *senha* e fornecer um meio de autenticação de usuários, de modo que sempre que um usuário solicitar um serviço, ele deve provar sua identidade.



Autenticação e Propagação de Identidade

Características do Kerberos

O Kerberos é o resultado de um esforço do MIT - Massachusetts Institute of Technology, conhecido como "Projeto Athena", iniciado em 1983. É um protocolo de autenticação de computadores em rede de código aberto, que fornece Login Único (SSO) baseado em

© Copyright Tecnisys 2026. All rights reserved.



um serviço de autenticação mútua confiável (usuários e serviços dependem de uma terceira parte: o servidor Kerberos). Foi adotado pela equipe Hadoop como componente para autenticação para acesso ao *Cluster* Hadoop. Dentre suas principais características, destacamos:

- **Confiabilidade**:
    - Os serviços com acesso controlado pelo protocolo só estarão disponíveis se o Kerberos também estiver. Utiliza uma arquitetura distribuída de servidores, com sistemas habilitados a fazer backup uns dos outros.
    - É um sistema de autenticação mútua que garante não apenas que o usuário é quem diz ser, mas também que os serviços que o usuário está acessando são os esperados. Usuários e o servidor sempre estarão seguros de que a contraparte com a qual estão interagindo é autêntica.
- **Escalabilidade**: Senhas ou chaves secretas são conhecidas apenas pelo Centro de Distribuição de Chaves (KDC) e pelos Principals do Kerberos (identidades únicas, como usuários ou serviços), tornando o sistema escalável para autenticar um grande número de entidades. Cada entidade precisa conhecer apenas sua própria chave secreta e registrá-la previamente no KDC.

- **Segurança**: A senha do usuário nunca é transmitida pela rede. O Kerberos utiliza tickets temporários (como o TGT e o Service Ticket), que são emitidos pelo servidor de autenticação (KDC) e têm tempo de vida limitado.

- **Transparência**: O usuário não tem conhecimento do processo de autenticação em si, exceto pela solicitação de uma senha.

- **Simplicidade**: Utiliza o sistema SSO (Single Sign-On), onde um único ticket pode ser usado por todos os serviços até que a validade expire.
  Simplifica a gestão de usuários: Criar, deletar, atualizar usuários no Kerberos é muito simples.

- **Velocidade**, pois utiliza Operações de Chave Simétrica, que são sempre mais rápidas em operações de autenticação SSL, que é baseada em chaves públicas e privadas.

- **Adaptabilidade**, pois integra-se facilmente com provedores de identidade corporativos, como Active Directory, FreeIPA ou sistemas baseados em LDAP.

© Copyright Tecnisys 2026. All rights reserved.

Arquitetura do Kerberos

O Kerberos opera no modelo cliente-servidor. Seu funcionamento básico consiste em:

- **Um *ticket***, que é um tipo de certificado, que informa de forma segura a identidade do usuário para quem o acesso foi originalmente concedido.
- **Um autenticador**, que é uma credencial gerada pelo cliente com informações que serão comparadas com o ticket, garantindo assim que o cliente que está apresentando o ticket é o mesmo para quem o ticket foi concedido.
- **Um centro de distribuição de chaves**, que fornece tickets válidos temporariamente ao usuário para acessar uma aplicação, que serão ratificados pelo autenticador. A aplicação examina o ticket e o autenticador quanto à validade e concede o acesso caso sejam válidos.

Para autenticar e verificar a autenticidade de consumidores, o Kerberos utiliza criptografia de chave simétrica (a mesma chave é usada para cifrar e decifrar os dados) e um componente central chamado KDC (Key Distribution Center), que mantém um banco de dados com todas as chaves secretas. Esse processo envolve três partes principais:

- **Authentication Server (AS) – realiza a autenticação inicial do usuário e emite o Ticket Granting Ticket (TGT).
- **Ticket Granting Server (TGS) – emite tickets de acesso para serviços específicos, com base no TGT previamente obtido. Isso evita que a senha do usuário precise ser reutilizada.
- **Kerberos Database – armazena as chaves secretas e identidades de todos os usuários e serviços autorizados no sistema.

A ideia que norteia a solução é a existência de um servidor capaz de entregar *tickets* ao usuário para acessar serviços. Esses *tickets* permanecem válidos por um determinado período. O serviço não precisa consultar o KDC para validar o ticket, pois consegue validá-lo localmente utilizando a chave secreta que compartilha com o KDC:

- O cliente solicita um *ticket* ao servidor Kerberos.
- O cliente submete o *ticket* ao serviço desejado e é autenticado.

> **NOTE**

© Copyright Tecnisys 2026. All rights reserved.



> Não existe a possibilidade de falsificar uma identidade ou forjar/reutilizar um *ticket*.

Autenticação de Serviços

Os serviços se autenticam com o Kerberos quando são iniciados, através do Kerberos Principal (uma identidade única que pode receber *tickets* para autenticação) e da *keytab* (que contém as credenciais de autenticação de recursos do *cluster*).

O Kerberos Principal autentica o serviço através da chave na *keytab*.

Após a autenticação, o KDC emite o *ticket*, que será inserido no conjunto de credenciais privadas. O serviço pode então atender ao cliente.

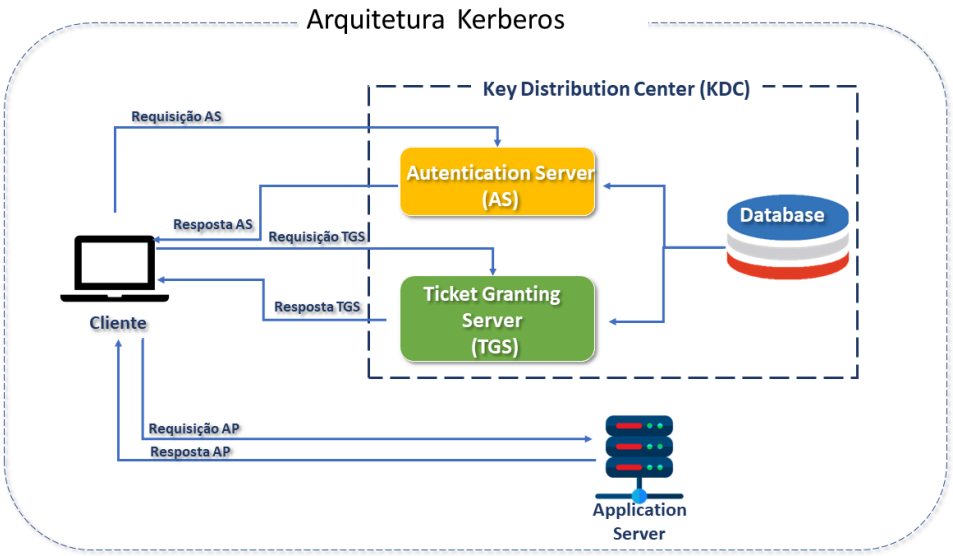
O processo de autenticação do usuário ocorre, resumidamente, da seguinte forma:

- O usuário se autentica usando seu Kerberos Principal.

> **NOTE**
>
> Inicialmente, o usuário deve fazer *login* na máquina cliente que está habilitada a se comunicar com o *Cluster* Hadoop.

- O usuário executa o comando `kinit` com o Kerberos Principal e a senha.
- `Kinit` autentica o usuário no KDC, obtendo o ticket resultante e colocando-o no cache de tickets no sistema de arquivos.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura Kerberos

Melhores Práticas para o Kerberos

- As configurações de criptografia no Kerberos geralmente são definidas para uma variedade de tipos, incluindo escolhas "fracas" como DES, por padrão. É recomendável remover os tipos fracos para garantir a melhor segurança possível.

- Ao usar AES-256, as extensões criptográficas Java precisam ser instaladas em todos os nós do *cluster* para permitir tipos de criptografia de "força ilimitada". É importante notar que alguns países proíbem o uso desses tipos de criptografia.

- Em ambientes onde os usuários do *Active Directory* (AD) precisam acessar Serviços Hadoop, é recomendável estabelecer confiança unidirecional entre Hadoop Kerberos e o Domínio AD.

- Como o Kerberos é um protocolo sensível ao tempo, todos os hosts no domínio devem ser sincronizados pelo tempo, por exemplo, usando o Network Time Protocol (NTP). Se a hora do sistema local de um cliente diferir daquela do KDC em apenas 5 minutos (o padrão), o cliente não poderá se autenticar.

Detalhes do Projeto Kerberos

O Kerberos foi desenvolvido em C.

Fontes:

© Copyright Tecnisys 2026. All rights reserved.

- Portal do MIT
- Wikipedia

© Copyright Tecnisys 2026. All rights reserved.

Apache Knox

Segurança de Perímetro

A Segurança de perímetro refere-se às barreiras naturais ou "construídas" para manter intrusos fora ou cativos dentro dos limites de nossas soluções.

Auxilia na proteção dos recursos do *Cluster* e fornece um único ponto de acesso para todas as interações REST e HTTP, simplificando a interação com o cliente.

Dentre seus principais benefícios, podemos citar:

- "Oculta" URL/portas específicas, agindo como um proxy.
- Simplifica a autenticação de vários serviços e UIs.
- Habilita terminação SSL no perímetro.
- Facilita o gerenciamento de *endpoints*.
- Fornece *logs* de acesso detalhados.

A segurança de perímetro típica inclui tecnologias como *proxies*, *firewalls*, sistemas de detecção de intrusão (IDS) e servidores virtuais de rede privada (VPN). Uma combinação disso tudo fornece uma segurança reforçada.



© Copyright Tecnisys 2026. All rights reserved.

Características do Knox

O Apache Knox atua como um tipo de proxy de aplicativo, dentro da camada de perímetro, onde recebe requisições destinadas a outro servidor e atua em nome do *cliente* para obter o recurso solicitado.

É um sistema criado para estender e simplificar o alcance dos serviços Apache Hadoop a usuários fora do *Cluster*, sem reduzir a segurança do Ecossistema. Foi projetado como um proxy reverso (servidor que reside na frente de um ou mais servidores web, interceptando solicitações dos clientes, com o objetivo de aumentar a segurança, confiabilidade e desempenho).

Apache Knox integra-se aos sistemas de gerenciamento de identidade e SSO (login único) e permite que a identidade desses sistemas seja usada para acesso aos *Clusters* Hadoop.

Knox entrega três grupos de serviços voltados ao usuário:

- **Serviços de Proxy:** Por meio de proxy de recursos HTTP.
- **Serviços de Autenticação:** Autenticando para acesso à API REST, assim como fluxo WebSSO para UIs. LDAP/AD, PreAuth baseado em cabeçalho, Kerberos, SAML e OAuth são opções.
- **Serviços de Cliente:** Usando scripts por meio de DSL ou classes do *Knox Shell* diretamente como SDK. O ambiente de script interativo KnoxShell combina o shell interativo do *Groovy Shell* com as *Classes Knox Shell SDK* para uma interação com os dados do *Cluster* Hadoop implantado.

Benefícios do Knox

As principais vantagens do Gateway Knox são:

- **Acesso simplificado:** O Knox estende os serviços REST/HTTP do Hadoop ao encapsular o Kerberos dentro do *Cluster*.
- **Segurança aumentada:** Expõe os serviços REST/HTTP do Hadoop sem revelar os detalhes da rede, fornecendo SSL/TLS *out-of-the-box* (pronto para uso).

© Copyright Tecnisys 2026. All rights reserved.



O Knox delegará solicitações externas dos clientes Hadoop aos serviços correspondentes e, antes de delegar, fornece todos os serviços de segurança configurados no *Cluster*:

- **Controle centralizado:** Reforça a segurança da API REST "centralizadamente", roteando solicitações para vários *Clusters* Hadoop.
- **Integração corporativa:** Suporta LDAP, Active Directory, SSO, SAML e outros sistemas de autenticação.
- **LDAP de demonstração:** Disponível por padrão no Apache Knox.
- **Registro de auditoria.**

Arquitetura do Apache Knox

O Apache Knox age como um ponto único de contato para serviços do Apache Hadoop no *Cluster*.

É executado como um *Cluster* servidor, na zona DMZ (zona desmilitarizada - situada entre uma rede confiável e uma rede não confiável, provendo isolamento físico entre as duas), isolando o *Cluster* Hadoop do resto da rede corporativa.

Sua principal característica é o provimento de um perímetro de segurança para APIs REST Hadoop, restringindo o número de *endpoints* de rede requeridos para acessar o *Cluster* Hadoop.

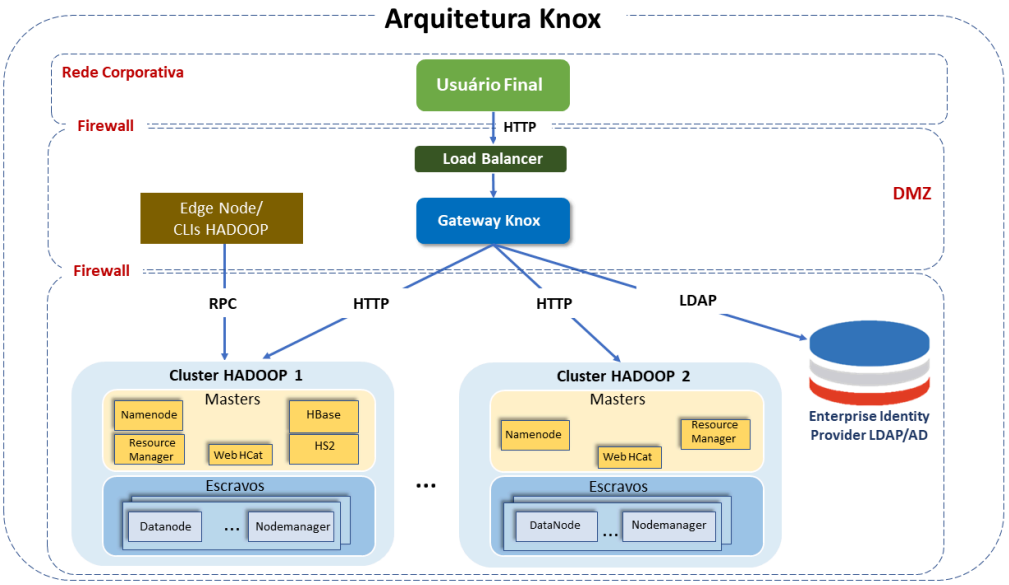
Como resultado, ele "esconde" a topologia do *Cluster* Hadoop. No perímetro da rede, provê um ponto único de autenticação e verificação de *tokens*.

Knox pode ser usado tanto com *Clusters Kerberizados* (protegidos por Kerberos) quanto *não Kerberizados*.

Em uma solução empresarial que emprega *Clusters* protegidos por Kerberos, provê segurança que integra-se bem com soluções de gestão de identidade da empresa, e, como já dito anteriormente, protegendo os detalhes da implementação do *Cluster* Hadoop e simplificando o número de serviços com os quais um cliente precisa interagir.

A arquitetura de implantação do Gateway Knox pode ser entendida no seguinte diagrama:

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura Knox

Como funciona o Apache Knox

O Gateway Apache Knox é um Gateway aplicativo para interação com as APIs REST e as UIs das implantações Apache Hadoop.

Fornece um único ponto de acesso para interações REST e HTTP com *Clusters* Apache Hadoop. Para isto, oferece três grupos de serviços voltados ao usuário:

- **Serviços de Proxy:** Fornece acesso ao Apache Hadoop por meio de proxy de recursos HTTP.
- **Serviços de Autenticação:** Autenticação para acesso à API REST, bem como fluxo WebSSO para UIs. LDAP/AD, Header based PreAuth, Kerberos, SAML, OAuth são todas opções disponíveis.
- **Serviços Cliente DSL/SDK:** O desenvolvimento cliente pode ser feito com *scripting* por meio de DSL ou classes Knox Shell diretamente como SDK. O ambiente de *script* interativo KnoxShell combina o Shell interativo do *groovy shell* com as classes Knox Shell SDK para uma interação com dados a partir do *Cluster* Hadoop implantado.

O Gateway Knox foi projetado como um proxy reverso com consideração para *plugabilidade* nas áreas de *policy enforcement* (imposição de políticas), por meio dos provedores e serviços de back-end para os quais ele faz proxy de solicitações.

A imposição de políticas abrange regras de autenticação/federação, autorização, auditoria, despacho, mapeamento de host e reescrita de conteúdo. A política é aplicada

© Copyright Tecnisys 2026. All rights reserved.

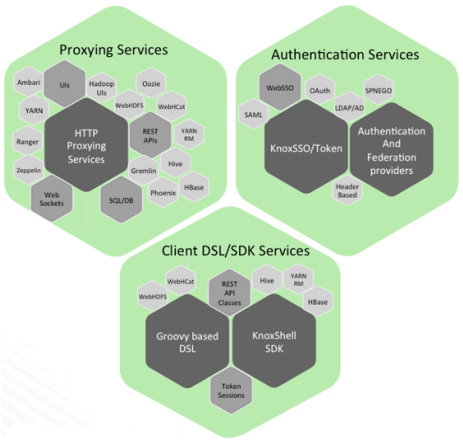
por meio de uma cadeia de provedores que são definidos no *topology deployment descriptor* (descritor de implementação de topologias) para cada *Cluster* Hadoop controlado pelo Knox. A definição de *Cluster* também é feita no descritor de implantação de topologia e fornece ao Gateway Knox o layout do *Cluster* para o propósito de roteamento e conversão entre URLs voltados ao usuário e internos do *Cluster*.

Cada *Cluster* Hadoop protegido pelo Knox tem seu conjunto de API REST representado por um único caminho de contexto de aplicativo específico do *Cluster*. Isso permite que o Knox proteja vários *Clusters* e apresente ao *consumidor* da API REST um único *endpoint* para acesso.

Com a gravação do descritor de implantação de topologia no diretório de topologias da instalação Knox, uma nova definição de *Cluster* Hadoop é processada, os provedores de *imposição de política* são configurados e o caminho de contexto do aplicativo é disponibilizado para uso pelos consumidores de APIs.

O Apache Knox também complementa muito bem a proteção de *Clusters* feita por *Kerberos*, fornecendo o isolamento de rede adequado e ainda:

- integrando-se bem com soluções de gerenciamento de identidade empresarial.
- protegendo os detalhes da implantação do *Cluster*(*hosts*, portas).
- simplificando o número de serviços com os quais os clientes precisam interagir.



Serviços do Knox

Recursos do Apache Knox

© Copyright Tecnisys 2026. All rights reserved.



- **Configuração para novos serviços e UIs:** Apache Knox provê um método direcionado de configuração para adicionar novos serviços de roteamento. Isto habilita para que novas API REST Hadoop sejam incorporadas rapidamente e facilmente. Esta funcionalidade foi adicionada na release 0.6.0.

- **Homepage:** O Apache Knox provê uma HomePage que pode ser usada como "front-door" de implementações e recursos que sejam publicados para acesso por meio dele. É uma boa alternativa para distribuir um link para a interface administrativa e obter *Quick Links*.

- **Autenticação:** Os provedores com função de autenticação são responsáveis por coletar as credenciais apresentadas pelo *consumidor* da API, validá-las e comunicar o sucesso ou falha da autenticação ao cliente ou ao restante da cadeia de provedores. O Knox traz consigo o provedor de autenticação *Shiro*, que aproveita o projeto Apache Shiro para autenticar credenciais BASIC em um repositório de usuários LDAP. Há suporte para OpenLDAP, Apache DS e Microsoft Active Directory.

- **Federação/SSO:** Para clientes que exigem que as credenciais sejam apresentadas a um conjunto limitado de entidades confiáveis dentro da empresa, o Knox pode ser configurado para federar a identidade autenticada de um evento de autenticação externa. Isto é feito por meio de provedores com a função de federação. O conjunto de provedores de federação prontos para uso incluem:

    - **IDP baseado em formulário padrão KnoxSSO:** A configuração padrão do KnoxSSO fornece um mecanismo de autenticação baseado em formulário que aproveita a autenticação Shiro para autenticar no LDAP/AD com credenciais coletadas de um desafio *form-based*.

    - **PAC4J:** O provedor pac4j adiciona recursos de autenticação e federação como SAML, CAS, OpenID Connect, Google, Twitter, etc.

    - **HeaderPreAuth:** Um mecanismo de propagação da identidade por meio de Headers HTTP que especifica o username e grupo para o usuário autenticado. Isto tem sido usado por use cases como SiteMinder e IBM Tivoli Access Manager.

- **Knox SSO:** O serviço Knox SSO é um serviço de integração que provê um *token* SSO normalizado para representar o usuário autenticado.
  Este token é geralmente usado para recursos WebSSO para UIs participantes e seu

© Copyright Tecnisys 2026. All rights reserved.

consumo de REST APIs Hadoop.
O Knox SSO abstrai a integração real do provedor de identidade dos aplicativos participantes para que estes precisem apenas estar "cientes" do *cookie* KnoxSSO.
O *token* é apresentado pelo navegador como um *cookie* e os aplicativos que participam da integração KnoxSSO podem validar criptograficamente o *token* apresentado e permanecer independentes da integração SSO subjacente.

- **Autorização:** A função de autorização é usada por provedores que tomam decisões de acesso para recursos solicitados com base no contexto de identidade do usuário efetivo.
  Este contexto de identidade é determinado pelo provedor de autenticação e pelas regras de mapeamento do provedor de asserção de identidade.
  A avaliação do usuário de contexto de identidade e *group principals* em relação a um conjunto de políticas de acesso é feita pelo provedor de autorização para determinar se o acesso deve ser concedido ao usuário para o recurso solicitado.

- **Auditoria:** A auditoria permite determinar quais ações foram executadas por quem durante um determinado período de tempo.
  A instalação é construída em uma extensão da estrutura Log4j e pode ser estendida substituindo a implementação "pronta para uso" por outra.

Recomendações e Boas práticas

- A integração do Knox com Apache Ranger é recomendada para checar as permissões dos usuários que desejam acessar os recursos do *Cluster*.
- Habilitar SSL é altamente recomendável. No caso de conexões Hive desconectadas via Apache Knox, recomendamos modificar o tempo limite da conexão e o número máximo de conexões no arquivo de configurações do site do gateway.
- Para melhorar os tempos de resposta por meio da configuração do Apache Knox, configure as propriedades abaixo, no *gateway.site*: `gateway.metrics.enabled=false`, `gateway.jmx.metrics.reporting.enabled=false`, `gateway.graphite.metrics.reporting.enabled=false`.

Serviços Suportados pelo Knox

Os seguintes serviços do Ecossistema Hadoop possuem integração com o Gateway Knox: (estes serviços podem ser consultados na página da comunidade - item "Supported Apache Hadoop Services" / "Supported Apache Hadoop ecosystem UIs")

Componentes suportados por Knox

© Copyright Tecnisys 2026. All rights reserved.

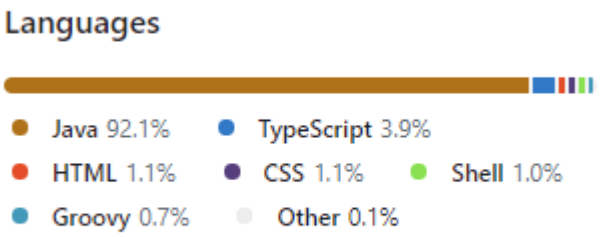
Componente	SSO	Proxy (API)	Proxy (UI)
Ambari	SIM	SIM	SIM
Métricas Ambari/Grafana			
Atlas	SIM	SIM	SIM
HBase		SIM	
HDFS			SIM
Hive(via JDBC)		SIM	
Hive(via WebHCat)		SIM	
MapReduce2	SIM		SIM
Zookeeper	SIM	SIM	SIM
Spark2/Spark History Server	SIM		SIM
WebHCat		SIM	
WebHDFS		SIM	
YARN	SIM	SIM	SIM
Zeppelin	SIM	SIM	SIM

TDP v2 - Knox - Serviços Suportados pelo Knox com Proxy e SSO (*Clusters Kerberizados ou não*)

Detalhes do Projeto Apache Knox

© Copyright Tecnisys 2026. All rights reserved.

O Apache Knox foi desenvolvido em Java.



Linguagens do Knox

Fontes:

- Comunidade Apache Knox
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Livy

Gerenciamento de Sessões Spark



Apache Livy é uma plataforma de serviço de código aberto que fornece uma maneira fácil de interagir com um *cluster* Apache Spark por meio de uma API REST. É essencialmente uma ponte entre aplicações e o Apache Spark, permitindo que usuários e aplicativos submetam e gerenciem trabalhos Spark de forma remota e interativa. Livy foi projetado para simplificar o processo de envio de trabalhos Spark, gerenciando a complexidade e os detalhes de comunicação com o *cluster* Spark.

Livy é ideal para cenários que exigem submissão programática de trabalhos Spark, como em sistemas de processamento de dados em lote, análise de dados em grande escala e integrações em ambientes de ciência de dados.

O Apache Livy foi inicialmente desenvolvido pela Cloudera como uma solução para simplificar a interação com *clusters* Spark. Desde o início, o foco do Livy tem sido oferecer uma interface RESTful para facilitar a submissão e o gerenciamento de trabalhos Spark em *clusters* de dados, especialmente em ambientes em que a interação direta com o Spark é complexa.

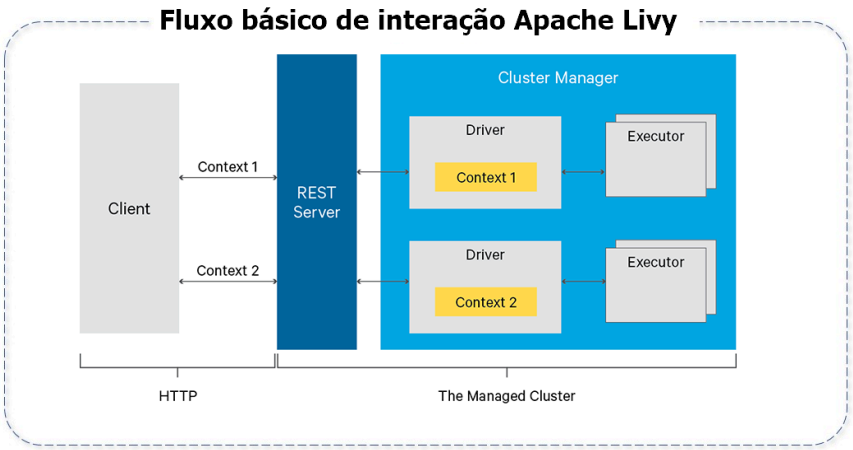
A evolução do Livy está alinhada com o desenvolvimento do Apache Spark, refletindo a necessidade crescente de ferramentas que facilitam o acesso a poderosos recursos de processamento de dados.

Características do Apache Livy

- **Submissão de Trabalhos Spark**: Permite a submissão e o gerenciamento de trabalhos Spark por meio de uma API REST;
- **Suporte a Múltiplas Linguagens**: Oferece suporte para trabalhos escritos em Scala, Python e R;
- **Gestão de Sessões**: Gerencia sessões Spark, facilitando a reutilização e otimizando o uso de recursos;
- **Segurança e Controle de Acesso**: Inclui recursos para garantir a segurança dos dados e do acesso ao *cluster*;

© Copyright Tecnisys 2026. All rights reserved.

- **Integração com Hadoop YARN**: Facilita a gestão de recursos em *clusters* por meio da integração com YARN.



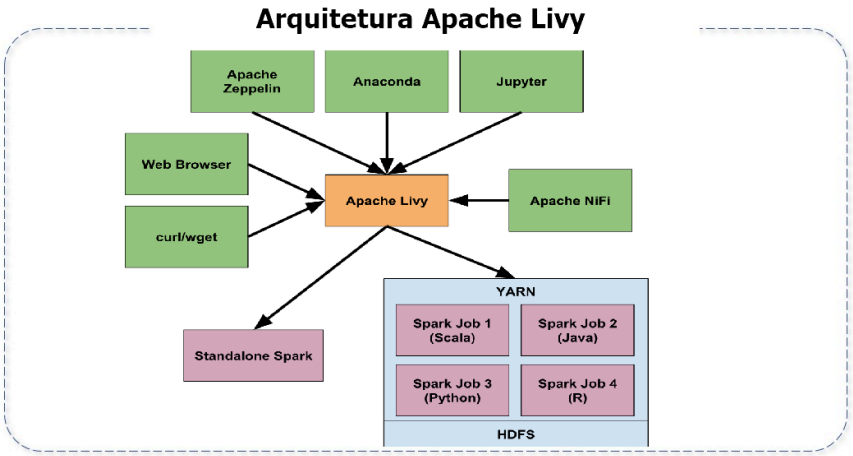
Fluxo Básico Apache Livy - submissão de _jobs_ por meio do Livy

Arquitetura do Apache Livy

Os principais componentes do Apache Livy são:

- **Servidor REST**: Fornece uma interface REST para a submissão e o gerenciamento de trabalhos Spark.
- **Gerenciador de Sessões**: Responsável por iniciar, manter e finalizar sessões Spark.
- **Integração com YARN**: Gerencia os recursos do *cluster* de forma eficiente.
- **Interface de Programação**: Fornece APIs para facilitar a submissão e o controle de trabalhos Spark.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura Apache Livy

Boas Práticas no Uso do Apache Livy

- **Gestão Eficiente de Sessões**: Administre as sessões Spark para otimizar o uso de recursos;
- **Segurança**: Implemente controles de segurança robustos;
- **Monitoramento**: Monitore o desempenho das sessões e do *cluster*;
- **Documentação e Integração**: Mantenha a documentação atualizada sobre a integração do Livy com outros sistemas;
- **Atualizações e Compatibilidade**: Mantenha-se informado sobre as novas versões do Livy e do Spark.

Quando Não Usar o Apache Livy

- **Ambientes de Baixa Latência**: O Livy pode não ser ideal para cenários que exigem baixíssima latência;
- **Trabalhos Simples e Isolados**: Para trabalhos Spark que não necessitam de integração complexa, o uso do Livy pode ser um exagero.

Detalhes sobre o projeto

O Apache Livy é desenvolvido principalmente em Java. Isso significa que uma compreensão de Java é benéfica para contribuir para o projeto ou integrá-lo de forma mais eficaz em sistemas existentes. Além disso, compreender os paradigmas de programação Java pode ajudar a otimizar a interação com o Livy e a implementar personalizações específicas conforme necessário.

© Copyright Tecnisys 2026. All rights reserved.

Fonte(s):

- Livy.apache.org

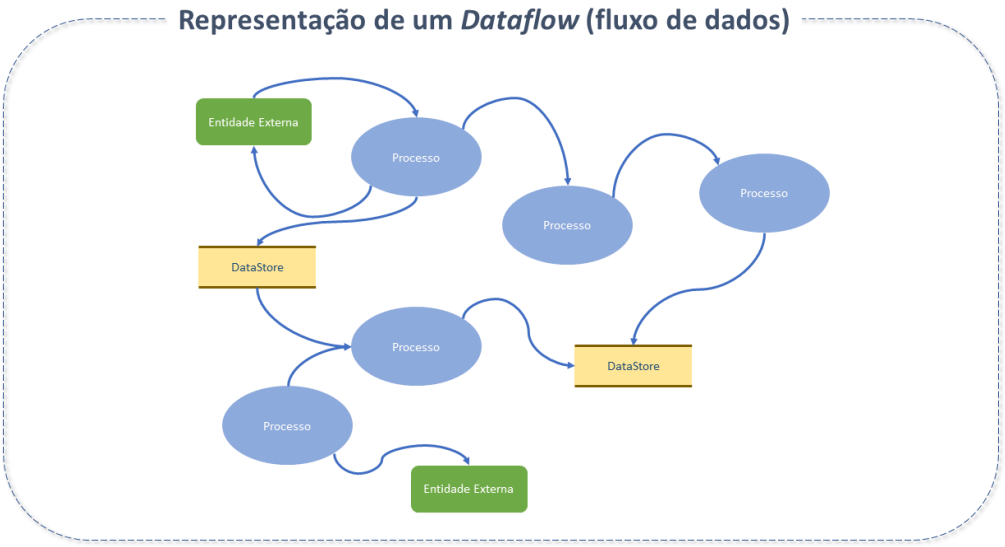
© Copyright Tecnisys 2026. All rights reserved.

Apache NiFi

Automação de Fluxo de Dados

Automação de fluxos de dados é um paradigma de software baseado na ideia de computação como um gráfico direcionado, representando o fluxo automatizado e gerenciado da informação entre sistemas.

A automação de fluxos de dados auxilia na captura, construção e colaboração em escala por meio de ferramentas automatizadas, garantindo maior eficiência dos sistemas por meio de relatórios otimizados e controláveis.



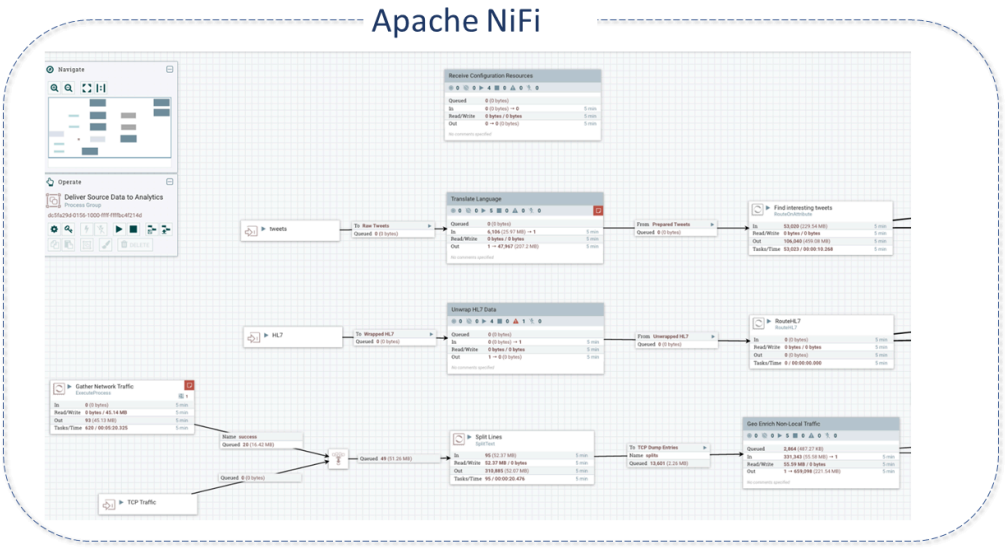
Representação de um Fluxo de dados

Características do Apache NiFi

O NiFi foi construído para automatizar fluxos de dados ("Dataflows"), suportar e atender o nível de rigor necessário para alcançar conformidade, privacidade e segurança na troca de dados entre sistemas.

NiFi tem se tornado especialmente valioso para as soluções de ponta que têm surgido ao longo dos últimos anos, para as quais o *Dataflow* é vital, como o SOA (Service Oriented Architecture), APIs (Application Protocol Integration), IOT (Internet of things) e Big Data.

© Copyright Tecnisys 2026. All rights reserved.



Interface do NiFi

Os conceitos de design fundamentais do Apache NiFi relacionam-se intimamente com as ideias de Programação baseada em Fluxos Flow Based Programming - fbp.

Termo NiFi	Termo FBP	Descrição
FlowFile	Informação do Pacote	O *FlowFile* representa cada objeto movido entre/para os sistemas.
Processador *FlowFile*	Caixa preta (Black Box)	São os Processadores que executam, de fato, o trabalho.
Connection	*Bounded Buffer*	Conexões provêem a ligação real entre processadores.

© Copyright Tecnisys 2026. All rights reserved.

Termo NiFi	Termo FBP	Descrição
Flow Controller	*Scheduler*	Provê *threads* para as extensões serem executadas e gerencia o *schedule* de quando as extensões recebem recursos para execução.

Tabela D - NiFi - Conceitos Associados ao FBP

Este modelo de design torna a Plataforma NiFi muito efetiva para construção de *dataflows* poderosos e escaláveis, habilitando:

- A criação visual e gerenciamento de gráficos direcionados (com sentidos associados à cada aresta) de processadores.
- Alta taxa de transferência e um *buffer* de dados natural, apesar das flutuações no processamento e nas taxas de fluxo.
- Modelos altamente concorrentes, com a capacidade de executar várias tarefas distintas simultaneamente (ou aparentemente simultâneas), permitindo que o desenvolvedor se despreocupe das complexidades típicas deste tema.
- O Desenvolvimento de componentes coesos, com baixo acoplamento, reutilizáveis em outros contextos e unidades testáveis.
- A Simplificação de funções críticas de dados, como contrapressão ("back-pressure") e "pressure release", tornando-se naturais e intuitivas (por meio do recurso *constrained connections*).
- O Tratamento de erros de forma tão natural quanto o "caminho feliz".
- Fácil compreensão e rastreamento de pontos onde o dado "entra" e "sai" do sistema.

Arquitetura do Apache NiFi

O NiFi é executado dentro de uma máquina virtual JVM, em um sistema operacional *host*. Seus principais componentes na JVM são:

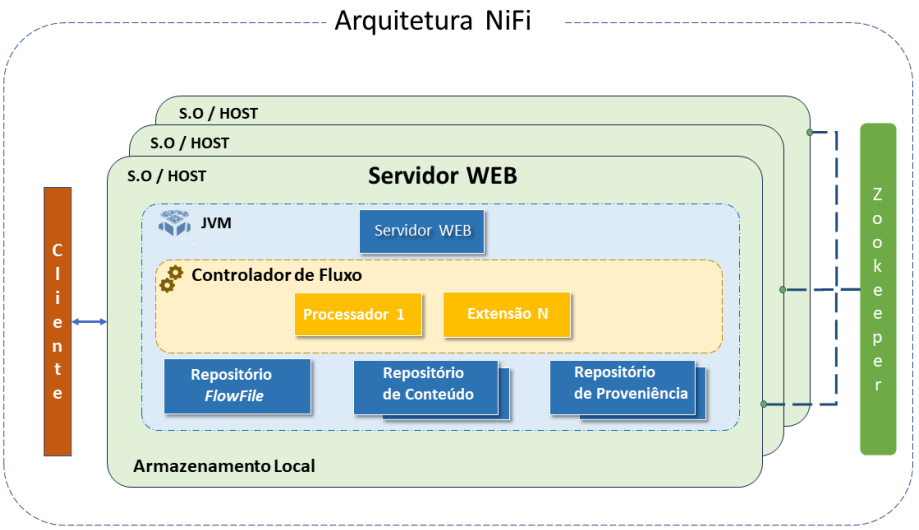
- **Web Server (Servidor Web)**: executa o controle visual e monitora os eventos. Hospeda o comando *HTTP-based* e a API de Controle.

© Copyright Tecnisys 2026. All rights reserved.



- **Flow Controller (Controlador de Fluxos)**: o "cérebro" da operação. Provê *threads* para as extensões serem executadas e gerencia o *schedule* de quando as extensões recebem recursos para execução.
- **Extensions (Extensões)**: Plugins que permitem a interação entre o NiFi e outros Sistemas.
  O NiFi provê vários pontos de "extensão" para que desenvolvedores possam adicionar funcionalidades. Todos operam e executam dentro da JVM:
  - **Processor (processador)**: A interface *processador* provê acesso aos *FlowFiles*, seus atributos e conteúdos.
  - **ReportingTask**: Provê métricas, informações de monitoramento e do estado interno do NiFi para *endpoints* como arquivos de *log*, email e *web services* remotos.
  - **ParameterProvider**: Provê parâmetros para uso por fontes externas.
  - **ControllerService**: Fornece mecanismo para criar serviços compartilhados entre todos os Processadores, *ReportingTasks* e outros *ControllerServices* em uma única JVM.
  - **FlowFilePriorizer**: Fornece um mecanismo pelo qual os FlowFiles em uma fila podem ser priorizados ou classificados, para posterior processamento em uma ordem mais eficaz em caso de uso específico.
  - **AuthorityProvider**: Determina quais privilégios e funções, se houver, devem ser concedidos a um determinado usuário.
- **Repositório *FlowFile***: É onde NiFi mantém/rastreia o status do *FlowFile* ativo. A implementação do repositório é "plugável". O "default" é um *log write-ahead* persistente localizado em uma partição do disco específica.
- **Repositório de Conteúdo**: É onde estão são mantidos os dados "em trânsito". É "plugável". A abordagem default é um mecanismo simples, que armazena blocos de dados no *file system*. Mais de um local de armazenamento do *file system* pode ser especificado, visando obter diferentes partições envolvidas na redução da contenção em qualquer volume único.
- **Repositório de Proveniência**: Armazena todas as informaçes sobre a proveniência dos dados que circulam pelo sistema. O repositório é "plugável" com a implementação *default* sendo para usar um ou mais volumes de discos físicos. Dentro de cada local o dado de evento é indexado e habilitado para busca.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura do NiFi

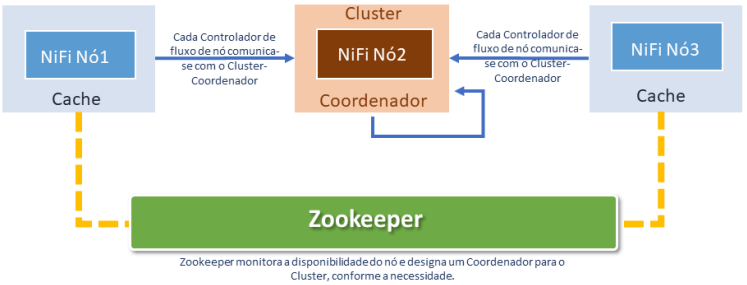
O NiFi também está habilitado a operar dentro de um *Cluster*.

Desde a versão NiFi 1.0, o NiFi emprega o paradigma de Zero-Leader: cada nó do *Cluster* NiFi executa as mesmas tarefas no dado, mas cada um operando em um conjunto distinto de dados.

O Apache Zookeeper elege um nó único como Coordenador. Todos os nós do *Cluster* reportam *heartbeat* e *status* para o Coordenador, que fica responsável por desconectar e conectar nós.

Adicionalmente, todo *Cluster* tem um Nó primário, também eleito pelo Zookeeper.

O *failover* é tratado automaticamente pelo Zookeeper.



Zero Leader Cluster

© Copyright Tecnisys 2026. All rights reserved.

Assim como o gerenciador de *Dataflow*, a interação com o *Cluster* NiFi pode se dar por meio de interface para o usuário (UI) de qualquer nó. Qualquer alteração é replicada para todos os nós no *Cluster*, permitindo múltiplos *entry-points*.

Expectativas de Performance

O Apache NiFi foi projetado para aproveitar totalmente os recursos do sistema *host* em que está operando. Essa maximização de recursos é particularmente forte em relação à CPU e disco.

I/O:

As taxas de transferência ou latência podem variar em função da configuração do sistema. Considerando que há abordagens "plugáveis" para a maior parte dos subsistemas NiFi, a performance dependerá da implementação. Considere usar as implementações padrão prontas para uso.

> **TIP**
>
> Podemos assumir uma taxa de "*read/write*" de 50MB por segundo em discos RAID modestos dentro de um servidor comum. Para grandes classes de *dataflows*, o NiFi está habilitado a alcançar 100MB por segundo ou mais de taxa de transferência (*throughput*), pois é esperado um crescimento linear para cada partição física e repositório de conteúdo adicionado ao NiFi.

CPU:

O controlador de fluxos aloca e gerencia *threads* para os processadores. Age como um "engine" ditando quando um processador receberá a *"thread*" para executar.

Além disso, permite adicionar serviços de controlador, que facilitam o gerenciamento de recursos, como conexões de Banco de Dados ou credenciais de provedor de serviços em nuvem.

Os serviços do controlador são *daemons*(executados em segundo plano) e fornecem configuração, recursos e parâmetros para os processadores executarem.

> **TIP**

© Copyright Tecnisys 2026. All rights reserved.



> O número ideal de *threads* a serem usados depende dos recursos do sistema "*host*" em termos de números de núcleos, se esse sistema está executando outros serviços e a natureza do processamento no fluxo. Para fluxos de I/O pesado é razoável disponibilizar muitas dezenas de *threads*.

RAM:

NiFi reside na JVM e é limitado pelo espaço de memória fornecido por ela. A coleta de lixo da JVM torna-se um fator muito importante tanto para restringir o tamanho total do *heap* quanto para otimizar a execução da aplicação ao longo do tempo.

> **TIP**
>
> *Jobs* NiFi podem ser intensivos em I/O quando lendo o mesmo conteúdo regularmente. Configure um disco grande o suficiente para otimizar a performance.

Recursos mais interessantes do Apache NiFi

NiFi Registry

É um subprojeto do Apache NiFi - uma aplicação complementar que provê "localização centralizada" para armazenamento e gestão de recursos compartilhados por uma ou mais instâncias do NiFi ou MiNiFi.

Disponibiliza os seguintes recursos:

- Implementação de *Flow Registry* para armazenar e gerenciar fluxos versionados.
- Integração com NiFi para permitir armazenamento, retenção e atualização de fluxos versionados do Flow Registry.
- Administração do Registry para definição de usuários, grupos e políticas.

A primeira implementação do Registry oferece suporte a fluxos versionados. Fluxos de dados em nível de grupo de processo criados em NiFi podem ser colocados sob controle de versão e armazenados em um registro. O registro organiza onde os fluxos são armazenados e gerencia as permissões para acessá-los, criá-los, modificá-los ou excluí-los.

© Copyright Tecnisys 2026. All rights reserved.

A interface de Usuário(UI) do NiFi Registry exibe os recursos compartilhados disponíveis e fornece mecanismos para criar e administrar usuários/grupos, *buckets* e políticas). Após o NiFi Registry instalado, um navegador da Web compatível pode ser utilizado para visualizar a UI.

> **NOTE**
>
> *Buckets* são containers que armazenam e organizam itens versionados, como fluxos e *bundles*(artefatos binários contendo uma ou mais extensões que podem ser executadas em NiFi ou MiNiFi).

MiNiFi

É um subprojeto do Apache NiFi - abordagem de coleta de dados complementar que suplementa os princípios básicos do NiFi no gerenciamento de fluxo de dados, com foco na coleta de dados na origem da sua criação.

O Apache NiFi MiNiFi fornece os seguintes recursos:

- Tamanho pequeno e baixo consumo de recursos.
- Gestão centralizada de agentes.
- Geração de proveniência de dados com toda a cadeia de custódia da informação.
- Integração com NiFi para gerenciamento de fluxo de dados subsequente.

Gerenciamento de Fluxos

- **Entrega Garantida**: A filosofia principal do NiFi é que, mesmo em altíssima escala, a entrega garantida é a regra.
  Isto é alcançado pelo uso eficaz do WAL (*log* de gravação antecipada) e repositório de conteúdo persistentes.
  Juntos eles são desenhados para permitir altas taxas de transações, distribuição de carga efetiva, *copy-on-write* e para "jogar" com as forças dos tradicionais *read-writes* no disco.
- **Data Buffering com Back Pressure e Pressure Release**: NiFi suporta o armazenamento em *buffer* de todos os dados da fila e tem a capacidade de fornecer *back pressure* àquelas filas que atingiram os limites ou para "envelhecer" os dados quando atingem uma idade especificada (seu valor pereceu).
- **Enfileiramento Priorizado**: NiFi permite configurar um ou mais esquemas de priorização para a recuperação de dados de uma fila. O default é "primeiro o mais

© Copyright Tecnisys 2026. All rights reserved.

antigo" mas há momentos em que dados "mais novos", maiores, ou algum outro esquema personalizado são a regra.
- **Flow Specific QoS**: (latência vs throughput, baixa tolerância, etc.): Há pontos no *Dataflow* onde o dado é absolutamente crítico e intolerante a falhas. Há ainda momentos em que devem ser processados e entregues em segundos para gerar algum valor. NiFi habilita o refinamento da configuração de fluxo.

Facilidade de Uso

- **Comando e Controle Visual**: *Dataflows* podem se tornar complexos. Proporcionar uma boa expressão visual pode ajudar a reduzir a complexidade e identificar áreas que precisam ser simplificadas.
  NiFi habilita o estabelecimento visual de *Dataflows* em tempo real. Mudanças no *dataflow* entram em vigor imediatamente. As alterações são refinadas e isoladas nos componentes impactados.
  Não é necessário interromper um fluxo inteiro ou conjunto de fluxos apenas para realizar uma modificação específica.
- **Templates**: *Dataflows* tendem a ser altamente orientados a padrões e embora existam muitos caminhos diferentes para solucionar um problema, o compartilhamento de boas práticas por especialistas no assunto beneficia a todos, além de permitir que os próprios especialistas se beneficiem com a colaboração dos demais.
- **Proveniência de Dados**: NiFi registra, indexa e disponibiliza automaticamente dados de proveniência à medida que os objetos fluem pelo sistema, mesmo em *fan-in*, *fan out*, transformações e outros.
  Estas informações se tornam extremamente críticas no suporte à conformidade, solução de problemas, otimizações e outros cenários.
- **Recovery/gravação de buffer contínuo de histórico refinado**: O repositório de conteúdo NiFi é desenhado para agir como um *buffer* contínuo do *history*.
  Dado é removido do repositório de conteúdo apenas quando "envelhece" ou quando é necessário mais espaço.
  Isto combinado com capacidade de proveniência cria uma base incrivelmente útil para permitir o *"click-to-content"*, *download* de conteúdo, reprodução, tudo num ponto específico do ciclo de vida de um objeto o qual pode abranger gerações.

Segurança

- **System to System**: Um *Dataflow* só é bom se for seguro. NiFi, em qualquer ponto do *Dataflow*, oferece troca segura com o uso de protocolos de criptografia como

© Copyright Tecnisys 2026. All rights reserved.

SSL bidirecional.
Adicionalmente, permite que o fluxo criptografe e descriptografe o conteúdo e use chaves compartilhadas ou outros mecanismos em ambos os lados da equação remetente/destinatário.

- **User to System**: NiFi permite a autenticação SSL bidirecional e provê autorização "plugável" para controlar o acesso pelo usuário em níveis específicos (somente leitura, gerenciador de fluxo de dados, administrador).
  Se um usuário inserir uma propriedade confidencial (como uma senha) no fluxo, ela será imediatamente criptografada no lado do servidor e nunca mais exposta no lado do cliente, mesmo em seu formato criptografado.
- **Autorização *multi-locatário***: O nível de autoridade de um *Dataflow* é aplicado a cada componente, viabilizando que usuário admin tenha um nível refinado de controle de acesso.
  Significa que cada cluster NiFi é capaz de lidar com os requisitos de uma ou mais organizações.
  Comparada a topologias isoladas, a autorização multi-locatário permite um modelo de autoserviço para gestão do *dataflow* e que cada time ou organização gerencie fluxos com total consciência do restante do fluxo, para os quais não têm acesso.

Arquitetura Extensível

- **Extensão**: NiFi é construído para extensão, e como tal, é uma plataforma na qual os processos de *Dataflow* podem ser executados e interagir de maneira repetitiva e previsível.
- **Isolamento do Carregador de Classe (*classloader*)**: Para qualquer sistema baseado em componentes, problemas de dependência podem ocorrer.
  NiFi endereça isto provendo um modelo de carga de classe personalizado, garantindo que cada pacote de extensão seja exposto a um conjunto muito limitado de dependências.
  Como resultado, extensões podem ser construídas sem se preocupar se podem entrar em conflito com outra extensão.
  O conceito destes pacotes de extensão é chamado "arquivos NiFi".
- **Protocolo de Comunicação Site-to-Site**: O protocolo de comunicação preferencial entre instâncias NiFi é o Protocolo NiFi Site-to-Site (s2s).
  S2S facilita a transferência de dados de uma instância NiFi para outra de forma eficiente e segura.
  As bibliotecas do cliente NiFi podem ser facilmente construídas e agrupadas em outros aplicativos ou dispositivos para se comunicar com o NiFi via S2S.

© Copyright Tecnisys 2026. All rights reserved.



Tanto o protocolo baseado em *socket* quanto o HTTP(s) são suportados no S2S como o protocolo de transporte subjacente, tornando possível incorporar um servidor *proxy* na comunicação S2S.

Modelo Flexível de Escala

- **Scale-out (Expansão / Clustering)**: NiFi é desenhado para escalabilidade horizontal por meio do uso de *clustering* de vários nós.
  Se um único nó for provisionado e configurado para lidar com centenas de MB por segundo, um *Cluster* modesto pode ser configurado para lidar com GB por segundo. Isto traz desafios de balanceamento de carga e *failover* entre NiFi e os sistemas dos quais obtem dados.
  O uso de protocolos baseados em filas assíncronas, como serviços de mensagem, kafka, etc. pode ajudar.
  O recurso *site-to-site* também é muito eficaz, pois é um protocolo que permite que o NiFi e um cliente (incluindo outro *Cluster*) conversem entre si, compartilhem informações sobre carregamento e troquem dados por portas autorizadas específicas.
- **Scale-up e down**: NiFi é também desenhado para *scale-up* e *scale-down* de uma maneira muito flexível.
  Em termos de "throughput", do ponto de vista da estrutura NiFi, é possível aumentar o número de tarefas simultâneas, no processador, na guia agendamento, durante a configuração.
  Isso permite que mais processos sejam executados simultaneamente, provendo grande *throughput*.
  Do outro lado do espectro, o NiFi pode ser dimensionado para execução em dispositivos de ponta, onde os recursos de hardware são limitados.

Diferença entre Apache Airflow e Apache NiFi

Por natureza, o Airflow é uma estrutura de "orquestração" e não uma estrutura de processamento de dados.

Enquanto o objetivo principal do NiFi está relacionado à categoria de "*stream processing*" (processamento de fluxos), automatizando a transferência de dados entre dois sistemas, o Airflow está mais relacionado à "*Workflow manager*" (Gerenciamento de Workflows).

© Copyright Tecnisys 2026. All rights reserved.

Importante ressaltar que as duas ferramentas não são mutuamente exclusivas e ambas oferecem recursos interessantes que auxiliam na solução de *silos de dados* (repositórios de dados isolados dentro da organização, sob o controle de um departamento, por exemplo).

O NiFi é uma ferramenta perfeita para Big Data. Não há melhor escolha quando se trata do tipo de *pipeline* "configure e esqueça".

O Airflow, por outro lado, é perfeito para agendamento de tarefas específicas, configuração de dependências e gerenciamento de fluxos de trabalho programáticos.

Permite, com facilidade, a visualização de dependências, código, tarefas de gatilho, progresso, *logs* e *status* de sucesso de *pipelines de dados*.

Boas Práticas para o Apache NiFi

Ambientes Separados

- **Ambientes separados para desenvolvimento**: É essencial manter ambientes separados para desenvolvimento, teste e produção para garantir a integridade dos dados e a estabilidade do sistema.

Considere o Usuário

Um dos conceitos mais importantes no desenvolvimento de um "processador" ou qualquer outro componente é a experiência do usuário:

- A documentação sempre deve ser fornecida para que todos possam utilizar o componente com facilidade.
- Consistência (convenções de nomenclatura), simplicidade e clareza são princípios fundamentais para tornar esta experiência adequada.

Coesão e Reutilização

Para criar uma unidade única e coesa, desenvolvedores são tentados a combinar várias funções em um único processador.

Adotar a abordagem de formatar dados para um ponto de extremidade específico e, em seguida, enviá-los para este ponto no mesmo processador pode não ser vantajoso, pois:

© Copyright Tecnisys 2026. All rights reserved.



- Pode tornar o processador muito complexo.
- Se o processador não conseguir se comunicar com um serviço remoto, encaminhará os dados para um relacionamento *failure* e ficará responsável por realizar a tradução dos dados novamente, e novamente...
- Se tivermos 5 diferentes processadores traduzindo dados de entrada em novo formato antes de enviá-los, teremos uma grande quantidade de código duplicado. Se o esquema mudar, muitos processadores devem ser atualizados.
- Estes dados intermediários são descartados quando o processador termina de enviar para o serviço remoto. O formato intermediário pode ser útil para outros processadores.

> **NOTE**
>
> Para evitar estes problemas e tornar os processadores mais reutilizáveis, um processador deve sempre seguir o princípio: "fazer uma única coisa e fazê-lo bem".

Ele deve ser dividido em dois processadores separados: um para converter os dados do formato X para Y e outro para enviar dados ao recurso remoto.

Convenções de Nomenclatura

Para fornecer uma aparência consistente aos usuários, é aconselhável que os processadores mantenham as convenções de nomenclatura padrão:

- Processadores que extraem dados de um sistema remoto são denominados:
  - `Get<Service>` (pesquisam dados de fontes arbitrárias por meio de protocolo conhecido, como GetHTTP, GetFTP) ou
  - `Get<Protocol>` (pesquisam dados de um serviço conhecido como GetKafka).
- Processadores que enviam dados para um sistema remoto são denominados:
  - `Put<Service>` ou
  - `Put<Protocol>`.
- Nomes de relacionamentos são caixa baixa e usam espaço para delinear palavras.
- Nomes de propriedade devem usar palavras significativas, como o título de um livro.

Anotações de Comportamento do Processador

© Copyright Tecnisys 2026. All rights reserved.



Ao criar um processador, o desenvolvedor deve fornecer dicas para a estrutura sobre como utilizá-lo com mais eficiência.
Isto é feito aplicando *anotations* à classe do processador.
As anotações que podem ser aplicadas estão em três sub-pacotes de arquivos *org.apache.nifi.annotation*:

- **Subpacote *documentation***: Fornecem documentação ao usuário.
- **Subpacote *lifecycle***: Instruem a estrutura sobre quais métodos devem ser chamados no processador para responder aos eventos de ciclo de vida apropriados.
- **Subpacote *behavior***: Ajudam a estrutura a entender como interagir com o processador em termos de agendamento e comportamento geral.

As seguintes anotações do pacote podem ser usadas para modificar como a estrutura lidará com seu processador (para obter mais detalhes, clique aqui):

- **EventDriven**: instrui a estrutura de que o processador pode ser agendado usando a estratégia de agendamento orientada a eventos.
- **SideEffectFree**: indica que o processador não tem nenhum efeito colateral externo ao NiFi.
- **SupportsBatching**: indica que não há problema em a estrutura agrupar várias confirmações de ProcessSession em uma única confirmação.
- **TriggerSerially**: impede que o usuário agende mais de uma *thread* concorrente para executar o método *onTrigger* de uma vez.
- **PrimaryNodeOnly**: restringe a execução do processador apenas no *Primary Node*.
- **TriggerWhenAnyDestinationAvailable**: Indica que o processador deve ser executado se algum relacionamento estiver "disponível" mesmo se uma das filas estiver cheia.
- **TriggerWhenEmpty**: ignora o tamanho das filas de entrada e aciona o processador, independentemente de haver ou não dados em uma fila de entrada.
- **InputRequirement**: ao fornecer um valor por meio desta anotação (INPUT_REQUIRED, INPUT_ALLOWED ou INPUT_FORBIDDEN), a estrutura saberá quando deve ser invalidada ou se o usuário deve ou não ser capaz de estabelecer uma conexão com o processador.

Buffer de Dados

O NiFi fornece uma capacidade genérica de processamento de dados. Os dados podem estar em qualquer formato.

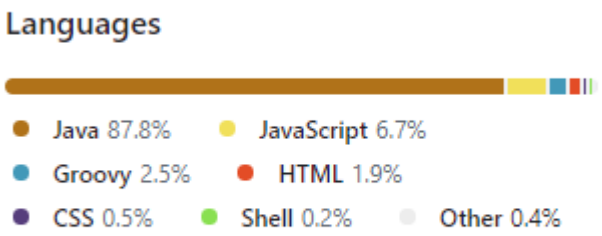
© Copyright Tecnisys 2026. All rights reserved.

Os processadores geralmente são escalonados com vários *threads*.
Um erro comum dos desenvolvedores é armazenar em *buffer* todo o conteúdo de um *FlowFile* na memória.
Salvo em casos em que é absolutamente necessário, isso deve ser evitado, a menos que se conheça o formato dos dados.

> **TIP**
>
> Em vez de armazenar esses dados em *buffer* na memória, é aconselhável avaliar os dados à medida que são transmitidos do Repositório de Conteúdo (ou seja, o conteúdo do *inputStream* fornecido pelo retorno de chamada do *ProcessSession.read*).

Detalhes do Projeto Apache NiFi

O Apache NiFi foi desenvolvido em Java.



Linguagens do NiFi

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **1.28.0**, distribuída pelo Helm Chart `tdp-nifi` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

© Copyright Tecnisys 2026. All rights reserved.

Fontes

- NiFi.apache.org
- NiFi.apache.org/registry
- cwiki.apache.org

© Copyright Tecnisys 2026. All rights reserved.

OpenMetadata

Recurso disponível somente a partir da versão 3.0.

Gerenciamento de Metadados



OpenMetadata é uma plataforma open-source unificada de gerenciamento de metadados criada para a descoberta de dados, governança, qualidade de dados, observabilidade e colaboração de pessoas.

O OpenMetadata é impulsionado por um repositório central de metadados, linhagem aprofundada e colaboração em equipe integrada.

É um dos projetos de código aberto de crescimento mais rápido, com uma comunidade vibrante e adoção por um conjunto diversificado de empresas em diversos setores.

Baseado em Padrões de Metadados Abertos e APIs, com suporte a conectores para uma ampla gama de serviços de dados, o OpenMetadata permite o gerenciamento de metadados de ponta a ponta.

A ideia central é que os metadados são um ativo de primeira classe e devem ser:

- Fáceis de consultar e explorar (descoberta).
- Confiáveis e atualizados (observabilidade e qualidade).
- Contextualizados (quem usa, para quê serve, em que fluxo se encaixa).
- Governados (com donos, políticas, classificações).

© Copyright Tecnisys 2026. All rights reserved.

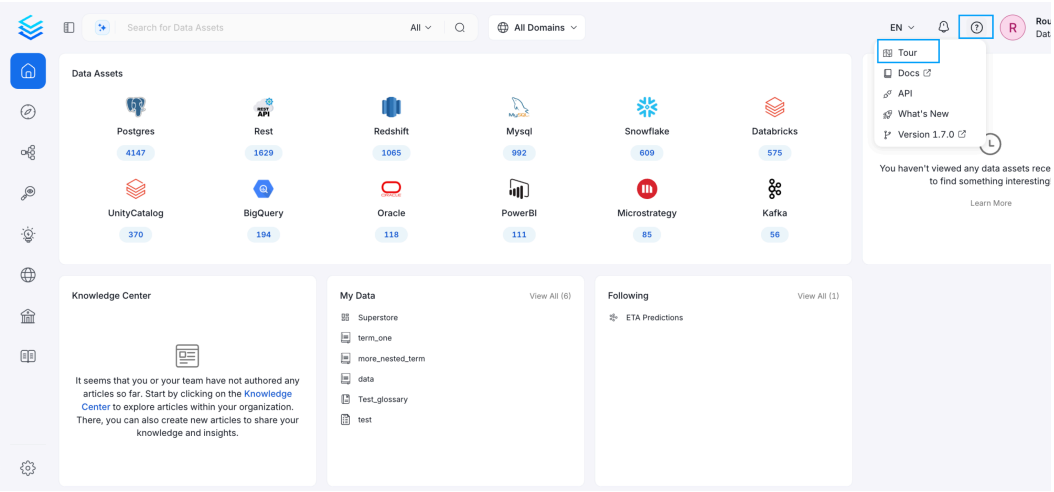


Figura 1 - OpenMetadata

O que ele faz

- **Catalogação e descoberta de ativos** (tabelas, dashboards, pipelines, modelos).
- **Documentação colaborativa**: descrições, comentários, dicionário de dados.
- **Governança e qualidade**: atribuição de donos, políticas, testes de dados integrados.
- **Linhagem visual**, inclusive em nível de coluna.
- **Observabilidade**: métricas de frescor, volume, qualidade, alertas.
- **Colaboração e notificações** via comentários, webhooks, controle de acesso.

Características do OpenMetadata:

- **Licenciado sob a Apache License, atualmente na versão 2.0**
- **Arquitetura simplificada**: essencialmente 4 serviços (API Server, DB relacional, Elasticsearch, Ingestion).
- **Framework de ingestão extensível**: + de 90 conectores prontos, e fácil criar novos.
- **Schema-first**: os contratos de metadados são JSON Schemas versionados.
- **APIs abertas**: toda funcionalidade exposta via REST, facilitando automação e integração com pipelines de CI/CD.

© Copyright Tecnisys 2026. All rights reserved.



- **UI moderna**: interface única para busca, visualização de linhagem, dashboards de qualidade e workflows de ingestão.

Arquitetura do OpenMetadata

O OpenMetadata segue uma arquitetura em camadas:

- **UI (Frontend)**: interface web em React, usada para exploração, colaboração e governança.
- **API Server (Backend)**: implementado em Java (Dropwizard/Jetty), expõe APIs REST e garante consistência.
- **Metadata Store** (DB relacional, geralmente MySQL/PostgreSQL): armazena entidades, relacionamentos e histórico.
- **Search Engine (Opensearch)**: indexa todos os ativos para busca rápida e facetada.
- **Ingestion Framework**: implementado em Python, roda como pipelines (pode ser agendado via Airflow ou standalone). Ele coleta metadados das fontes conectadas e envia para a API.

© Copyright Tecnisys 2026. All rights reserved.

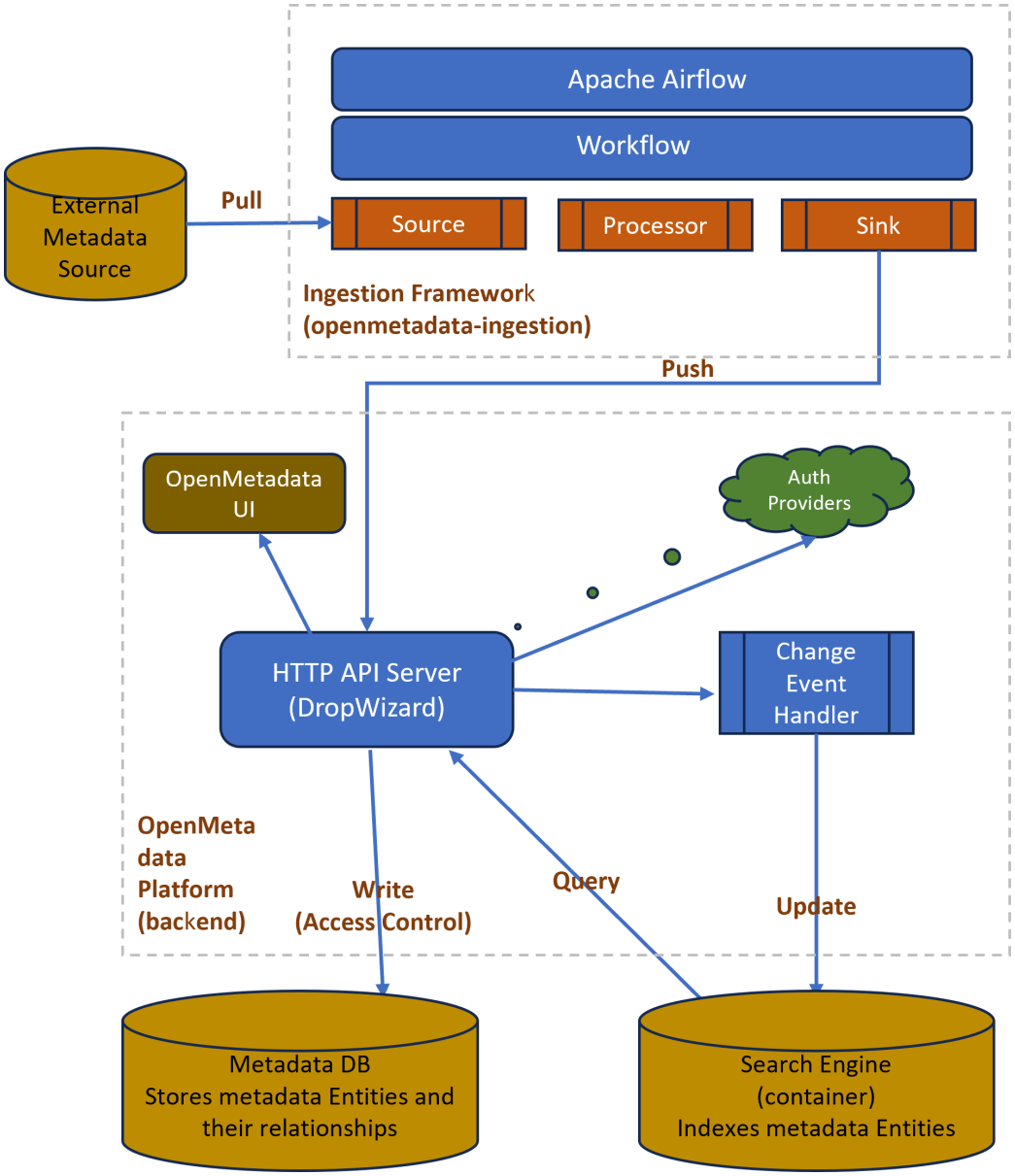


Figura 2 - Arquitetura do OpenMetadata

Seus principais componentes são:

- **Entities (Entidades)**: tabela, coluna, usuário, pipeline, dashboard etc.
- **Ingestion Workflows**: conectores para bancos, ETL, BI, ML.
- **Glossary/Taxonomy**: dicionário de termos de negócio e domínios.
- **Policies & Roles**: controle de acesso granular.
- **Lineage Graph**: motor que constrói e visualiza dependências.

© Copyright Tecnisys 2026. All rights reserved.

- **UI Colaborativa**: comentários, notificações, integração Slack/Jira.

Recursos do OpenMetadata

- **Conectores prontos para bancos** (Snowflake, BigQuery, Redshift, Hive), BI (Looker, Superset), pipelines (Airflow, Dagster, Spark).
- **Linhagem detalhada** com drill-down até colunas individuais.
- **Data Quality**: criação de testes nativos (ex.: valores esperados, limites, consistência).
- **Observabilidade**: dashboards de frescor, volume, popularidade.
- **Governança**: atribuição de donos, classificação de dados sensíveis, domínios organizacionais.
- **Extensibilidade**: fácil criar conectores e schemas customizados.

Boas Práticas no Uso do OpenMetadata

- **Comece pequeno**: catalogue fontes críticas e expanda gradualmente.
- **Automatize ingestão**: use workflows agendados (Airflow, cron).
- **Defina donos claros**: atribua ownership de tabelas, dashboards, pipelines.
- **Adote glossário corporativo**: termos de negócio ajudam não-técnicos a entender dados.
- **Monitore qualidade**: configure testes mínimos em dados críticos.
- **Integre com comunicação**: Slack, Teams, Jira para alertas e colaboração.
- **Versione e audite**: mantenha histórico de metadados e mudanças.
- **Use RBAC e políticas**: restrinja quem pode editar o quê.
- **Participe da comunidade**: aproveite extensões já prontas e contribua de volta.

Detalhes sobre o projeto

© Copyright Tecnisys 2026. All rights reserved.



OpenMetadata tem seu Backend desenvolvido em Java, usando framework Dropwizard/Jetty.

Implementa o Ingestion Framework em Python e o FrontEnd (UI) em TypeScript/React.

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **1.12.4**, distribuída pelo Helm Chart `tdp-openmetadata` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fonte(s):

- openmetadata.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone



Armazenamento de Objetos

Recurso disponível somente a partir da versão 2.3.

Ozone é um armazenamento de objetos redundante e distribuído otimizado para cargas de trabalho de Big Data. O principal ponto de design do ozone é a escalabilidade, e ele visa escalar para bilhões de objetos.

Ozone separa o gerenciamento de namespace e o gerenciamento de espaço de bloco; isso ajuda o ozone a escalar muito melhor. O namespace é gerenciado por um daemon chamado Ozone Manager (OM), e o espaço de bloco é gerenciado pelo Storage Container Manager (SCM).

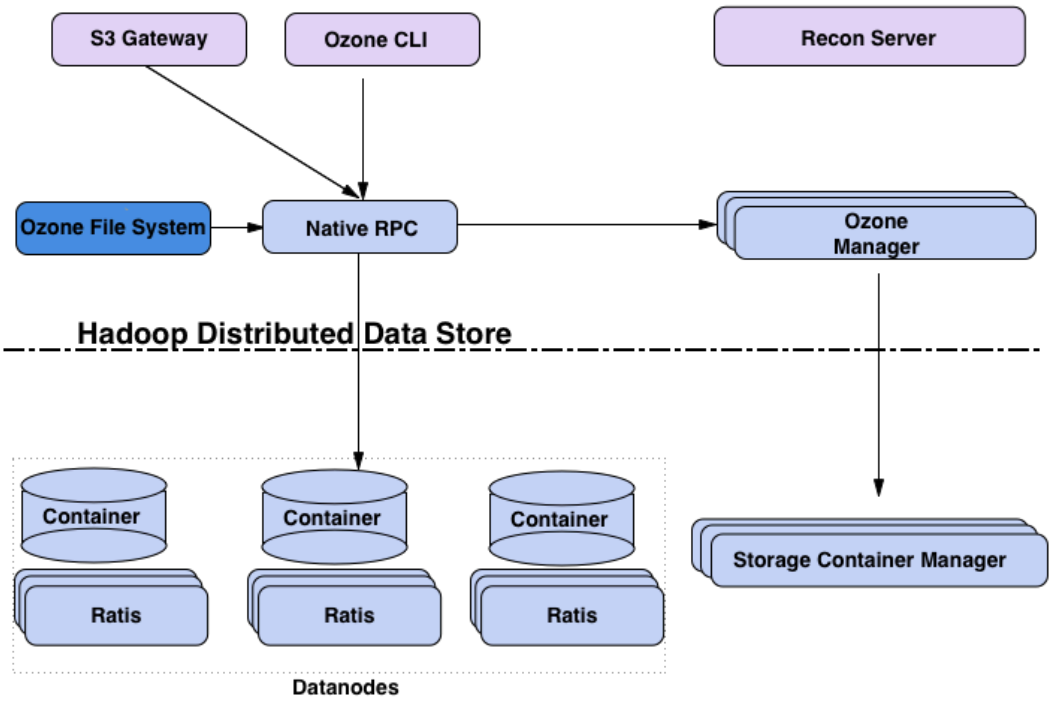
Ozone consiste em volumes, buckets e chaves. Um volume é semelhante a um diretório inicial no mundo do ozone. Somente um administrador pode criá-lo.

Os volumes são usados para armazenar buckets. Depois que um volume é criado, os usuários podem criar quantos buckets forem necessários. Ozone armazena dados como chaves que vivem dentro desses buckets.

O namespace do Ozone é composto por muitos volumes de armazenamento. Os volumes de armazenamento também são usados como base para a contabilidade de armazenamento.

O diagrama de blocos mostra os principais componentes do Ozone.

© Copyright Tecnisys 2026. All rights reserved.

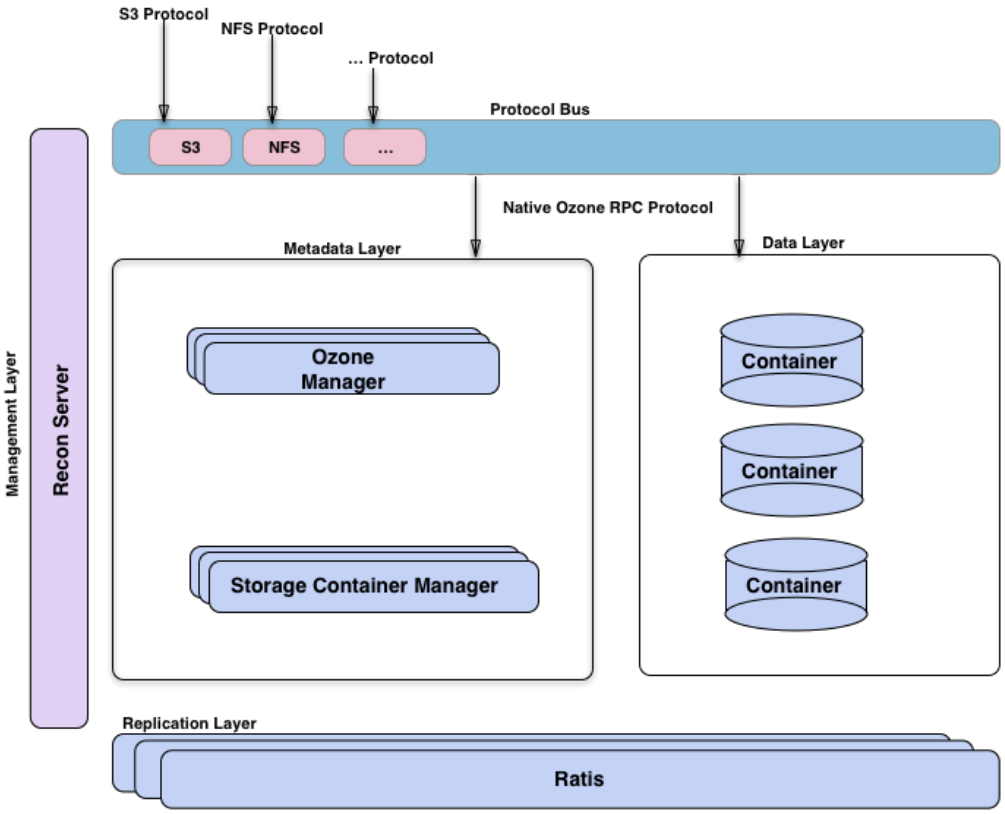


Armazenamento de dados - Ozone

O Ozone Manager é o gerenciador de namespace, o Storage Container Manager gerencia a camada física e de dados.

Diferentes Perspectivas

© Copyright Tecnisys 2026. All rights reserved.



Ozone Funcional

Qualquer sistema distribuído pode ser visto de diferentes perspectivas. Uma maneira de olhar para o Ozone é imaginá-lo como o Ozone Manager como um serviço de namespace construído sobre o HDDS, um armazenamento de bloco distribuído.

Outra maneira de visualizar o Ozone é olhar para as camadas funcionais; temos uma camada de gerenciamento de dados de metadados, composta pelo Ozone Manager e pelo Storage Container Manager.

Temos uma camada de armazenamento de dados, que são basicamente os nós de dados e eles são gerenciados pelo SCM.

A camada de replicação, fornecida pelo Apache Ratis, é usada para replicar metadados (OM e SCM) e também usada para consistência quando os dados são modificados nos nós de dados.

Temos um barramento de protocolo que permite que o Ozone seja estendido por meio de outros protocolos. Atualmente, temos apenas suporte ao protocolo S3 construído por meio do barramento de protocolo. O barramento de protocolo fornece uma noção

© Copyright Tecnisys 2026. All rights reserved.

genérica de que você pode implementar novos protocolos de sistema de arquivos ou armazenamento de objetos que chamam o protocolo O3 Native.

Arquitetura do Apache Ozone

A arquitetura do Apache Ozone foi projetada para gerenciar bilhões de objetos com escalabilidade e resiliência. Ele separa a gestão do namespace do gerenciamento de blocos, utilizando os seguintes componentes principais:

- **Ozone Manager (OM)**:
  - Gerencia o namespace (volumes, buckets e chaves).
  - Utiliza o protocolo Ratis para garantir consistência em clusters distribuídos.
- **Storage Container Manager (SCM)**:
  - Gerencia containers, que são a unidade de replicação.
  - Coordena o posicionamento e replicação dos dados entre os DataNodes.
- **DataNodes**:
  - Armazenam fisicamente os dados em containers.
  - Garantem alta disponibilidade e replicação eficiente.
- **Protocolo Ratis**:
  - Implementa consenso distribuído para garantir replicação consistente.

Fluxo de Operações no Ozone

- **Gravação de Dados**:
  - O cliente solicita ao OM a alocação de blocos nos DataNodes.
  - Os dados são gravados diretamente nos DataNodes e replicados conforme necessário.
- **Leitura de Dados**:
  - O cliente consulta o OM para localizar os dados.
  - Os dados são recuperados diretamente dos DataNodes.
- **Replicação de Dados**:
  - O SCM gerencia a replicação entre DataNodes para garantir tolerância a falhas.

© Copyright Tecnisys 2026. All rights reserved.

Expectativas de Performance

O Ozone emprega técnicas avançadas para alcançar alta performance, como:

- **Replicação via RAFT para containers abertos**.
- **Replicação assíncrona para containers fechados (dados frios)**.

Recursos do Apache Ozone

O Apache Ozone oferece um conjunto robusto de recursos para atender às demandas de armazenamento de dados massivos. Aqui estão os mais destacados:

- **Compatibilidade com S3**:

  APIs que permitem integração com sistemas baseados em nuvem. Ideal para migrações de dados e arquiteturas híbridas.

- **Integração com Hadoop**:

  Alternativa direta ao HDFS, sem necessidade de ajustes em frameworks como Apache Spark, Hive e YARN.

- **Alta Disponibilidade**:

  Baseado no Hadoop Distributed Data Store (HDDS), garante replicação consistente e tolerância a falhas.

- **Suporte Multimodal**:

  Permite armazenamento como sistema de arquivos ou objetos, dependendo do caso de uso.

- **Escalabilidade Horizontal**:

  Suporta bilhões de objetos, sendo ideal para aplicações em Big Data e Cloud Native.

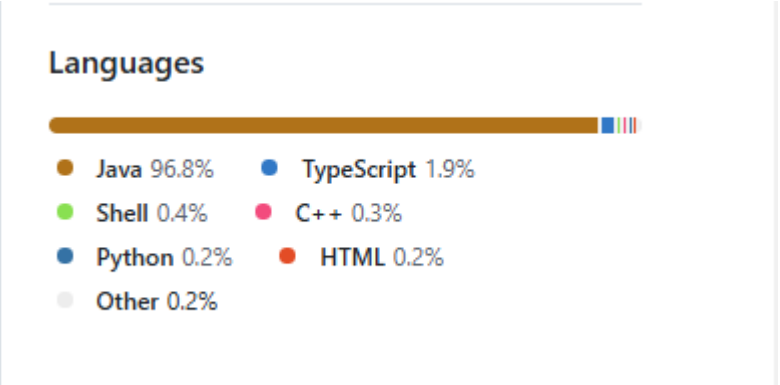
- **Reconhecimento de Topologia**:

  Otimiza pipelines de leitura e escrita com base no posicionamento dos nós no cluster.

© Copyright Tecnisys 2026. All rights reserved.

Detalhes do Projeto Apache Ozone

O Apache Ozone foi construído principalmente em Java, que é a linguagem base de todo o ecossistema Hadoop, incluindo HDFS, YARN e outros componentes relacionados. Essa escolha permite ao Ozone integrar-se perfeitamente ao Hadoop e aos frameworks associados, como Spark, Hive e MapReduce.



Linguagem do Apache Ozone

Fontes

Apache Software Foundation

© Copyright Tecnisys 2026. All rights reserved.

Apache Ranger



Segurança de Dados

Uma vez que a identidade do usuário tenha sido estabelecida por meio da autenticação, a segurança precisa garantir quais ações ou serviços aquela identidade pode acessar.

A segurança de dados refere-se às medidas de proteção empregadas para proteger informação digital contra acesso não autorizado, corrupção ou roubo, durante todo o seu ciclo de vida, preservando sua confidencialidade, integridade e disponibilidade.

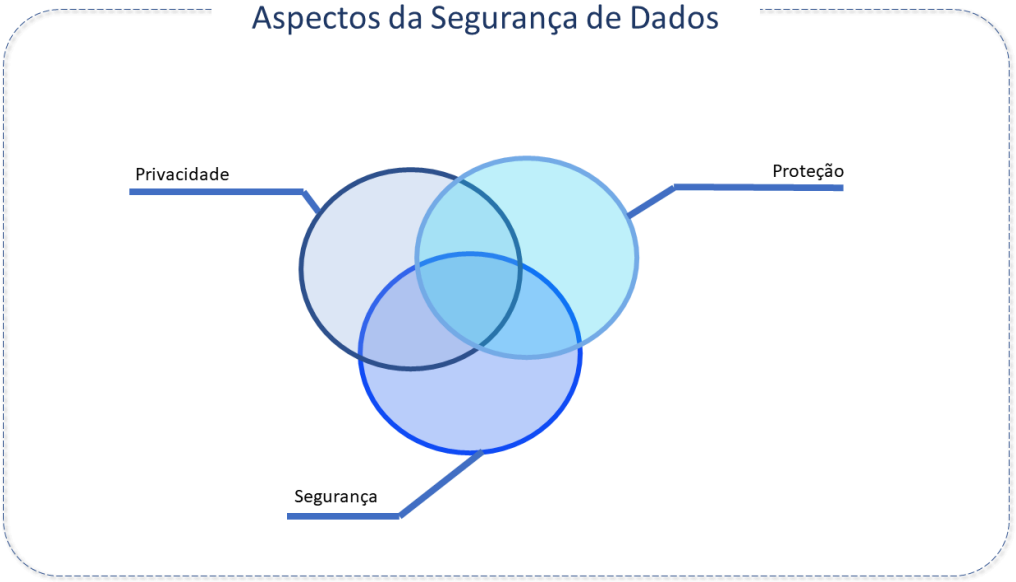
A transformação digital mudou profundamente os aspectos de como as empresas atuais operam e competem. O grande volume de dados que criam, manipulam e armazenam torna os ambientes mais complexos, expandindo a superfície de ataque e tornando-a mais difícil de monitorar e proteger.

Ao mesmo tempo, a crescente demanda pública por iniciativas de proteção de dados, com vários novos regulamentos de privacidade se junta às disposições de segurança de longa data. O valor dos dados nunca foi tão grande. A perda de segredos comerciais ou propriedade intelectual pode afetar a inovação e lucratividade.

Alia-se a tudo isto, os crescentes desafios inerentes à complexidade dos ambientes distribuídos e híbridos atuais.

Felizmente, existem ferramentas de **cibersegurança** disponíveis, cuja proteção pode reduzir os riscos à segurança quando usadas como parte da auditoria de segurança. Estas ferramentas devem conhecer onde os dados residem, rastrear quem tem acesso a eles e bloquear atividades de alto risco.

© Copyright Tecnisys 2026. All rights reserved.



Aspectos da Segurança de Dados

Características do Apache Ranger

Apache Ranger é um framework criado para habilitar, monitorar e gerenciar, de forma abrangente, a segurança de dados na Plataforma Hadoop.

Com a implementação do Apache YARN a Plataforma Hadoop passou a suportar uma arquitetura mais ampla de data-lake, possibilitando a execução de múltiplos *workloads* em um ambiente "multilocatário", o que, consequentemente, demandou uma estrutura de segurança de dados mais robusta, com suporte e monitoramento ao acesso de dados e administração centralizada das políticas de segurança.

O Apache Ranger foi criado com os seguintes objetivos:

- **Gestão centralizada da segurança:**
  Possibilitar o gerenciamento de todas as tarefas relacionadas a segurança em uma UI central ou usando REST APIs.

- **Autorização "refinada":**
  Para realizar ações e operações com componentes ou ferramentas Hadoop e gerenciamento por meio de ferramenta de administração centralizada.

- **Padronização dos métodos de autorização:**
  Em todos os componentes Hadoop.

© Copyright Tecnisys 2026. All rights reserved.

- **Aprimoramento do suporte a distintos métodos de autorização:**
  Controle de acesso baseado em papéis, controle de acesso baseado em atributos, etc.

- **Centralização da auditoria:**
  Centralização da auditoria do acesso pelo usuário e ações administrativas (relacionadas a segurança) em todos os componentes do Hadoop.

Arquitetura do Apache Ranger

Os principais componentes do Apache Ranger são:

- **Ranger Admin Server/Portal:**
  Todas as políticas são gerenciadas de forma centralizada por meio de um portal Web que atua como uma Interface central para gerenciamento da segurança e permite a definição de repositórios, criação e atualização de políticas de segurança, gerenciamento de usuários e grupos, definição de políticas de auditoria e controle e visualizar atividades de auditoria.
  O portal, por sua vez, possui três partes distintas:

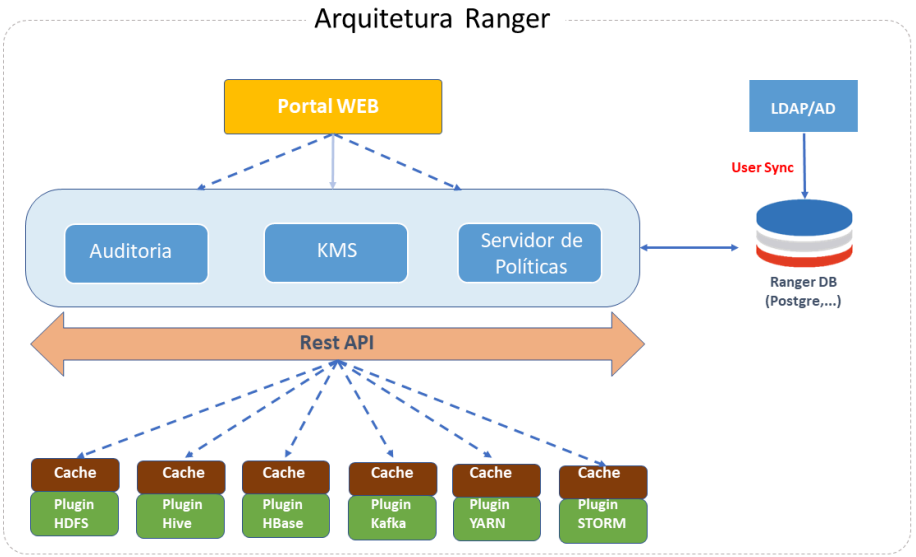
  - **Auditoria:**
    Monitora atividades do usuário em nível de recursos e pesquisa de log de auditoria de buscas baseado em alguns filtros.

  - **Gerenciador de Políticas:**
    Adiciona ou modifica políticas para grupos ou usuários.
    Especifica quais usuários são admins, quem pode acessar ou modificar políticas.

  - **KMS:**
    Usado para armazenar chaves usadas na criptografia de dados HDFS.

- **Plugins Ranger:**
  São programas Java incorporados aos processos dos componentes do *Clusters*. Estes Plugins obtêm as políticas de um servidor central e as armazenam localmente, atuando como módulo de autorização e avaliando as solicitações do usuário em relação às políticas de segurança obtidas. Além disso, envia esses dados ao servidor de auditoria.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura Ranger

Serviços Suportados pelo Apache Ranger

Atualmente são suportados os seguintes serviços:

Componente	S/N
HDFS	SIM
Hive	SIM
HBase	SIM
Storm	SIM
Knox	SIM
SolR	SIM
Kafka	SIM
YARN	SIM

© Copyright Tecnisys 2026. All rights reserved.

Componente	S/N
Ozone	SIM
Kudu	SIM
Kylin	SIM
NiFi	SIM
NiFi Registry	SIM
Sqoop *(obsoleto)*	SIM
Atlas	SIM
ElasticSearch	SIM
Presto	SIM
Schema Registry	SIM

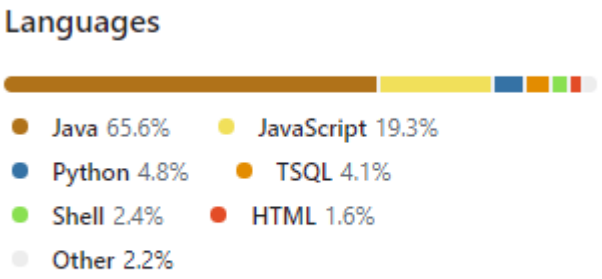
Boas práticas para o Apache Ranger:

- Para autorização HDFS, mude o *umask* HDFS de 022 para 077 para prevenir que qualquer novo arquivo ou pasta seja acessado por outra pessoa que não o *owner*.
- Auditoria no Apache Ranger pode ser controlada como uma política. Configure SSL para todos os plugins habilitados.
- Uma política *default* é criada quando o Apache Ranger é instalado via Ambari, para todos os arquivos e diretórios no HDFS e com auditoria habilitada. Ambari usa esta política para realizar *smoke test* via "Ambari QA" para verificar serviços HDFS. Uma política similar para habilitar auditoria entre todos os arquivos e pastas deve ser criada se administradores desabilitarem esta política *default*.

Detalhes do Projeto Apache Ranger

Apache Ranger foi desenvolvido em Java predominantemente.

© Copyright Tecnisys 2026. All rights reserved.



Linguagens do Ranger

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **2.7.0**, distribuída pelo Helm Chart `tdp-ranger` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes:

- Comunidade Ranger
- Wiki
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Ranger KMS



Gerenciador de Chaves Criptográficas

Um gerenciador de chaves criptográficas (KMS, do inglês "Key Management System") é uma solução ou serviço projetado para gerenciar e proteger as chaves de criptografia usadas na segurança de dados e comunicações. Eles desempenham um papel crucial na garantia da confidencialidade e integridade dos dados sensíveis.

A ideia de gerenciar chaves criptográficas remonta aos primórdios da criptografia, mas o termo "KMS" tornou-se mais relevante com o aumento do uso de tecnologias de criptografia em ambientes empresariais.

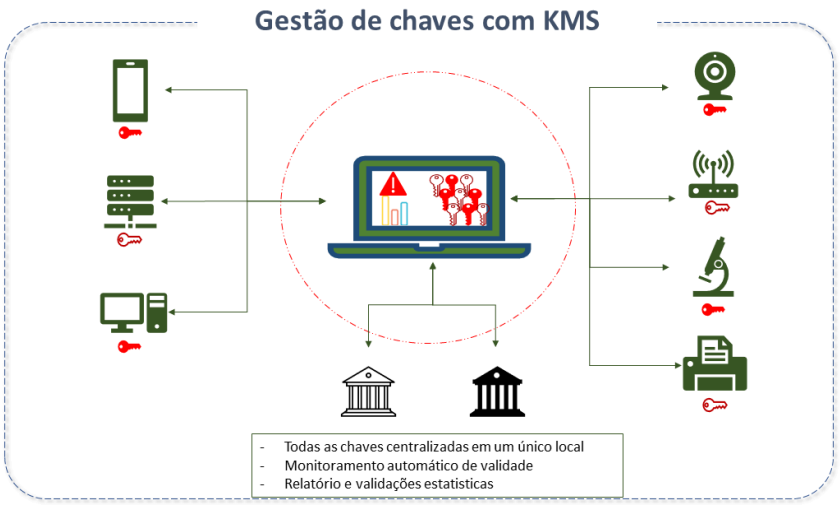
Dentre suas características relevantes, destacamos:

- **Armazenamento seguro de chaves:** KMSs mantêm chaves criptográficas em um ambiente altamente seguro, como Hardware Security Modules (HSMs) ou software criptografado;
- **Geração de chaves:** Eles podem criar chaves criptográficas fortes e aleatórias;
- **Distribuição de chaves:** KMSs distribuem chaves para sistemas e aplicativos que precisam delas de forma segura;
- **Controle de acesso:** Permitem um controle rigoroso sobre quem pode acessar e usar as chaves;
- **Auditoria:** Registram atividades relacionadas a chaves para fins de auditoria e conformidade;
- **Políticas de segurança:** Permitem a aplicação de políticas de segurança, como rotação de chaves e expiração de chaves.

A arquitetura de um KMS varia, mas geralmente consiste em um componente central de gerenciamento de chaves, um repositório seguro de chaves e mecanismos de controle de acesso. HSMs são frequentemente usados para armazenar chaves de forma

© Copyright Tecnisys 2026. All rights reserved.

segura, protegendo-as contra acesso não autorizado. As políticas de segurança são aplicadas para determinar quem pode acessar e usar as chaves.



Arquitetura básica de um KMs

Principais Tipos no Mercado:

- **HSMs (Hardware Security Modules):** Dispositivos físicos dedicados para armazenar e gerenciar chaves criptográficas de forma altamente segura;
- **KMS em Nuvem:** Serviços baseados em nuvem que oferecem gerenciamento de chaves como um serviço, como o AWS Key Management Service (KMS) da Amazon Web Services;
- **KMS Locais:** Soluções de software que podem ser implantadas em infraestruturas locais ou em ambientes de nuvem privada;
- **KMS de Terceiros:** Muitos fornecedores de segurança oferecem suas próprias soluções de KMS para uso em conjunto com seus produtos.

Os KMSs são essenciais para garantir a segurança de dados sensíveis, protegendo as chaves de criptografia contra ameaças internas e externas. Eles desempenham um papel fundamental em ambientes empresariais e são amplamente adotados em setores que requerem alto nível de segurança, como saúde, financeiro e governamental.

Apache Ranger Key Management Service (KMS)

O Ranger KMS é uma parte do ecossistema Apache Ranger que fornece gerenciamento de chaves e serviços de criptografia para proteger dados sensíveis em clusters Hadoop e sistemas de armazenamento relacionados. É uma ferramenta de código aberto projetada para simplificar o gerenciamento de chaves criptográficas e garantir a conformidade com políticas de segurança e regulamentações.

© Copyright Tecnisys 2026. All rights reserved.

O Ranger KMS é uma ferramenta importante para organizações que lidam com dados sensíveis em clusters Hadoop e sistemas de armazenamento distribuídos, ajudando a protegê-los por meio de gerenciamento de chaves criptográficas e aplicação de políticas de segurança. Desempenha um papel fundamental no ecossistema Apache Ranger e é amplamente utilizado em ambientes de Big Data.

Características do Ranger KMS

Dentre suas principais características, destacamos:

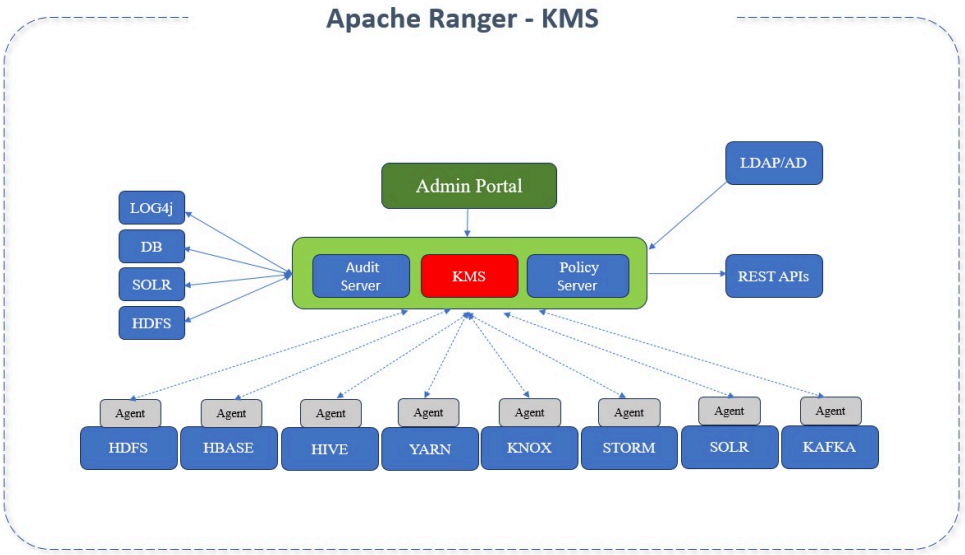
- **Gerenciamento seguro** de chaves criptográficas;
- **Integração com sistemas Hadoop** e outros sistemas de armazenamento;
- **Controle de acesso** baseado em políticas para chaves;
- **Auditoria e registro de atividades** relacionadas a chaves;
- **Suporte** para vários algoritmos de criptografia;
- **Integração com o Apache Ranger** para aplicação de políticas de segurança;
- **Extensibilidade** para suportar sistemas de armazenamento adicionais.

Arquitetura do Ranger KMS

A arquitetura do Ranger KMS inclui os seguintes componentes principais:

- **Key Management Service:** Serviço central que gerencia chaves criptográficas;
- **Key Database:** Armazena as chaves criptográficas de maneira segura;
- **Policy Engine:** Aplica políticas de segurança para controlar o acesso às chaves;
- **Audit Log:** Registra todas as atividades relacionadas a chaves para fins de auditoria;
- **Ranger Admin:** A interface de administração do Apache Ranger para configurar políticas de segurança.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura do Ranger KMS

Recursos do Ranger KMS

- **Segurança Avançada:** Reforça a segurança dos dados criptografando informações sensíveis;
- **Facilitação da Conformidade:** Simplifica o cumprimento das regulamentações de segurança de dados;
- **Controle Fino:** Fornece um controle mais detalhado sobre quem pode acessar e gerenciar chaves criptográficas;
- **Registro e Monitoração:** Facilita o registro e monitoração de atividades para fins de auditoria e conformidade;
- **Integração Facilitada:** Pode ser integrado de forma fluida com sistemas Hadoop e outras tecnologias de Big Data.

Boas Práticas para o Apache Ranger

Ao usar o Ranger KMS (Key Management Service) ou qualquer outro sistema de gerenciamento de chaves criptográficas, é essencial seguir boas práticas de segurança para garantir a proteção adequada das chaves e dos dados sensíveis que elas protegem. Aqui estão algumas boas práticas ao utilizar o Ranger KMS:

- **Políticas de Segurança Claras:** Defina políticas de segurança claras que descrevam quem tem acesso às chaves e sob quais condições.
- **Controle de Acesso Granular:** Para garantir que apenas as pessoas e os sistemas autorizados possam acessar as chaves criptográficas.

© Copyright Tecnisys 2026. All rights reserved.

- **Rotação Regular de Chaves:** Estabeleça políticas para a rotação regular de chaves, garantindo que as chaves não se tornem obsoletas e vulneráveis;
- **Auditoria de Atividades:** Habilite a auditoria de atividades no Ranger KMS para registrar e monitorizar todas as operações relacionadas às chaves;
- **Proteção Física e Lógica:** Mantenha um alto nível de proteção física e lógica em torno dos componentes do Ranger KMS, incluindo Hardware Security Modules (HSMs) se utilizados;
- **Atualizações e Patches:** Mantenha o Ranger KMS atualizado com as últimas atualizações de segurança e patches para corrigir vulnerabilidades conhecidas;
- **Segregação de Funções:** Implemente a segregação de funções para garantir que as responsabilidades de gerenciamento de chaves estejam distribuídas entre várias pessoas ou equipes;
- **Monitoração Contínua:** Estabeleça um sistema de monitoração contínua para detectar atividades suspeitas ou não autorizadas relacionadas às chaves;
- **Criptografia de Comunicação:** Garanta que todas as comunicações entre os componentes do Ranger KMS sejam criptografadas para proteger os dados em trânsito;
- **Treinamento e Conscientização:** Forneça treinamento adequado aos administradores e utilizadores do Ranger KMS para garantir o uso correto e seguro do sistema;
- **Backup e Recuperação:** Implemente políticas de *backup* e recuperação para garantir que as chaves não sejam perdidas em caso de falha do sistema;
- **Teste de Recuperação de Desastres:** Realize testes regulares de recuperação de desastres para garantir que você possa restaurar as chaves em caso de falha grave;
- **Conformidade com Regulamentações:** Certifique-se de que suas práticas de uso do Ranger KMS estejam em conformidade com as regulamentações de segurança de dados relevantes para sua organização e setor;
- **Revisão Periódica de Políticas:** Realize revisões periódicas das políticas de segurança e atualize-as conforme necessário para atender às necessidades em constante evolução de segurança;
- **Acesso Mínimo Necessário:** Siga o princípio do "acesso mínimo necessário", concedendo acesso apenas às chaves e informações criptográficas que os usuários ou sistemas precisam para desempenhar suas funções;
- **Gerenciamento de Incidentes:** Estabeleça procedimentos de gerenciamento de incidentes para responder efetivamente a qualquer violação de segurança ou evento suspeito;
- **Avaliação de Vulnerabilidades:** Realize avaliações regulares de vulnerabilidades para identificar e mitigar possíveis ameaças à segurança.

© Copyright Tecnisys 2026. All rights reserved.

Lembrando que as boas práticas de segurança devem ser adaptadas às necessidades específicas de sua organização e às regulamentações aplicáveis. É fundamental manter-se atualizado sobre as melhores práticas de segurança e ajustar suas políticas e procedimentos de acordo com as mudanças no cenário de ameaças.

Detalhes do Projeto Ranger KMS

O Ranger KMS (Key Management Service) é principalmente escrito na linguagem de programação Java. O ecossistema Apache, no qual o Apache Ranger faz parte, é amplamente baseado em Java, e isso se estende ao código-fonte do Ranger KMS. O uso do Java permite que o Ranger KMS seja executado em várias plataformas e sistemas operacionais compatíveis com Java, tornando-o altamente portátil e escalável.

Fontes(s):

- Comunidade Apache

© Copyright Tecnisys 2026. All rights reserved.

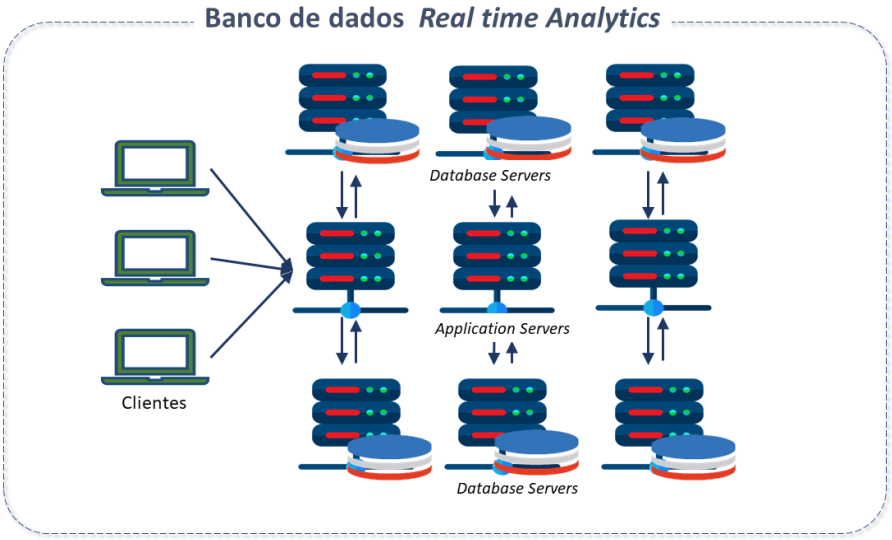
Apache Spark

Plataforma de Computação Distribuída

Com a evolução das redes de computadores, um novo paradigma computacional surgiu e se tornou extremamente poderoso: a possibilidade de distribuição do processamento entre computadores diferentes.

A computação distribuída permite a repartição e especialização das tarefas computacionais, segundo a natureza e função de cada computador.

Esse paradigma surgiu para solucionar um grande problema da computação, que é a necessidade de computadores com poder de processamento suficiente para realizar a análise do grande volume de dados disponível atualmente.



Computação Distribuída

Características do Apache Spark

O Apache Spark é um mecanismo de análise unificada para processamento de dados em larga escala em computação distribuída.

Criado em 2009 na Universidade da Califórnia pelo AMPLab de Berkeley, o Spark rapidamente ganhou uma grande comunidade, levando à sua adoção pela Apache

© Copyright Tecnisys 2026. All rights reserved.



Software Foundation em 2013.

Desenvolvedores de mais de 300 empresas auxiliaram na sua implementação, e uma vasta comunidade com mais de 1.200 desenvolvedores de centenas de organizações continuam a contribuir para seu contínuo refinamento.

É usado por organizações em uma gama de setores e sua comunidade de desenvolvedores é uma das maiores existentes.

O poder do Spark vem do seu processamento em memória (*in-memory*).

Ele usa um conjunto distribuído de nós com muita memória e codificação de dados compacta, juntamente com um planejador de consultas (*query planner*) otimizado para minimizar o tempo de execução e a demanda por memória.

Como realiza cálculos em memória (*in-memory*), pode processar até 100 vezes mais rápido do que as estruturas que processam no disco.

É ideal para processar grandes volumes de dados em Analytics, treinamento de modelos para aprendizado de máquina e IA.

Adicionalmente, o Spark executa uma pilha de bibliotecas nativas de aprendizado de máquina e processamento de gráficos e estruturas de dados semelhantes a SQL, permitindo um desempenho excepcional.

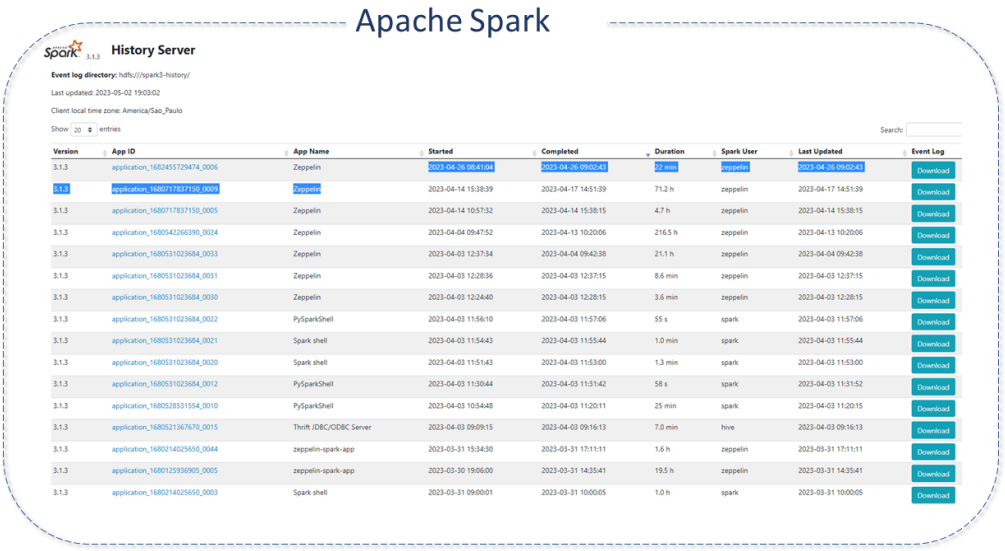
Conta com mais de 80 operadores de alto nível, facilitando a criação de aplicativos paralelos.

O Spark guarda algumas semelhanças com o Hadoop. Ambos são estruturas de código aberto para processamento de dados analíticos, vivem na Apache Software Foundation, contêm bibliotecas de aprendizado de máquina e podem ser programados em várias linguagens diferentes.

No entanto, o Spark estende o número de cálculos possíveis com o Hadoop, aprimorando o componente de processamento de dados nativo do Hadoop, o MapReduce.

O Spark usa a infraestrutura do Hadoop Distributed File System (HDFS), mas melhora suas funcionalidades e fornece ferramentas adicionais, como a implementação de aplicativos em um cluster Hadoop (com SIMR - Spark Inside MapReduce) ou YARN.

© Copyright Tecnisys 2026. All rights reserved.



Interface Apache Spark

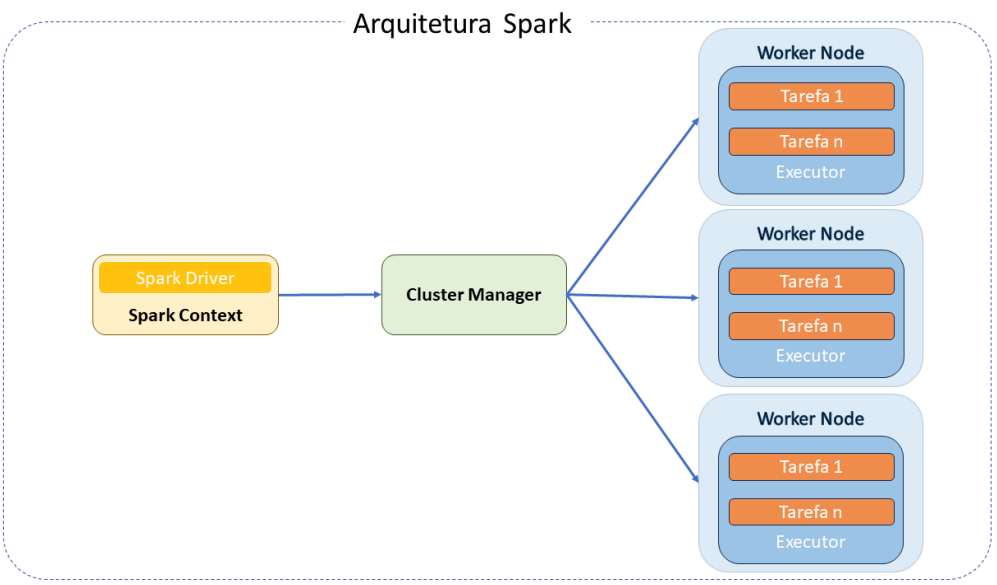
Arquitetura do Apache Spark

O Apache Spark é um mecanismo de processamento distribuído que opera no princípio de coordenador/trabalhador.

Sua arquitetura consiste nos seguintes componentes principais:

- **Spark Driver:** É o mestre da arquitetura Spark. É a aplicação principal que gerencia a criação e execução do processamento definido pelo programador.
- **Cluster Manager**: Um componente opcional necessário apenas se o Spark for executado de forma distribuída. Ele é responsável por gerenciar as máquinas que serão usadas como trabalhadores (*workers*).
- **Spark Workers:** São as máquinas que executam efetivamente as tarefas enviadas pelo programa Driver. Se o Spark for executado de forma local, a máquina pode desempenhar tanto o papel de Driver quanto de Worker.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura Spark

Componentes Fundamentais do Modelo de Programação Spark

- **Resilient Distributed Datasets (RDD)**: O principal objeto do modelo de programação Spark. É nesses objetos que os dados são processados. Eles armazenam os dados na memória para realizar várias operações como carregamento, transformação e ações (cálculos, gravações, filtros, uniões e map-reduce) nos dados. Abstraem um conjunto de objetos distribuídos no cluster, geralmente executados na memória principal. Podem estar armazenados em sistemas de arquivo tradicional, no HDFS e em alguns bancos de dados NoSQL, como o HBase.

- **Operações**: Representam transformações (agrupamentos, filtros, mapeamento de dados) ou ações (contagens e persistências) realizadas em um RDD. Um programa Spark normalmente é definido como uma sequência de transformações ou ações realizadas em um conjunto de dados.

- **Spark Context**: O contexto é o objeto que conecta o Spark ao programa que está sendo desenvolvido. Pode ser acessado como uma variável em um programa.

Bibliotecas do Apache Spark

Além das APIs, existem bibliotecas que compõem seu ecossistema e fornecem capacidades adicionais:

© Copyright Tecnisys 2026. All rights reserved.



- **Spark Streaming**: Pode ser usado para processar dados de streaming em tempo real com base na computação de microbatch. Para isso, utiliza o DStream, que é basicamente uma série de RDDs para processar os dados em tempo real. É escalável, possui alto *throughput*, tolerante a falhas e suporta workloads batch ou streaming. O Spark Streaming permite a leitura/gravação a partir/para tópicos Kafka nos formatos texto, csv, avro e json.

- **Spark SQL**: Fornece a capacidade de expor os conjuntos de dados Spark por meio de uma API JDBC. Isso permite a execução de consultas no estilo SQL sobre esses dados, fazendo uso de ferramentas tradicionais de BI e visualização. Além disso, permite a utilização de ETL para extração de dados em diferentes formatos (Json, Parquet ou Banco de dados), transformá-los e expô-los para consultas *ad-hoc*.

- **Spark MLlib**: A biblioteca de aprendizagem de máquina do Spark, que consiste em algoritmos de aprendizagem, incluindo classificação, regressão, clustering, filtragem colaborativa e redução de dimensionalidade.

- **Spark GraphX**: Uma nova API do Spark para grafos e computação paralela. Simplificando, estende os RDDs do Spark para grafos. Para apoiar a computação de grafos, expõe um conjunto de operadores fundamentais (subgrafos e vértices adjacentes), bem como uma variante otimizada do Pregel. Além disso, inclui uma crescente coleção de algoritmos para simplificar tarefas de análise de grafos.

- **Variáveis Compartilhadas**: O Spark oferece dois tipos de variáveis compartilhadas para torná-lo mais eficiente na execução em clusters:

    - **Broadcast:** São variáveis somente leitura (*read-only*) que são armazenadas em cache em todos os nós do cluster para acesso ou uso por tarefas. Em vez de enviar os dados junto com cada tarefa, o Spark distribui as variáveis de transmissão (*broadcast*) na máquina usando algoritmos eficientes de transmissão para reduzir os custos de comunicação.

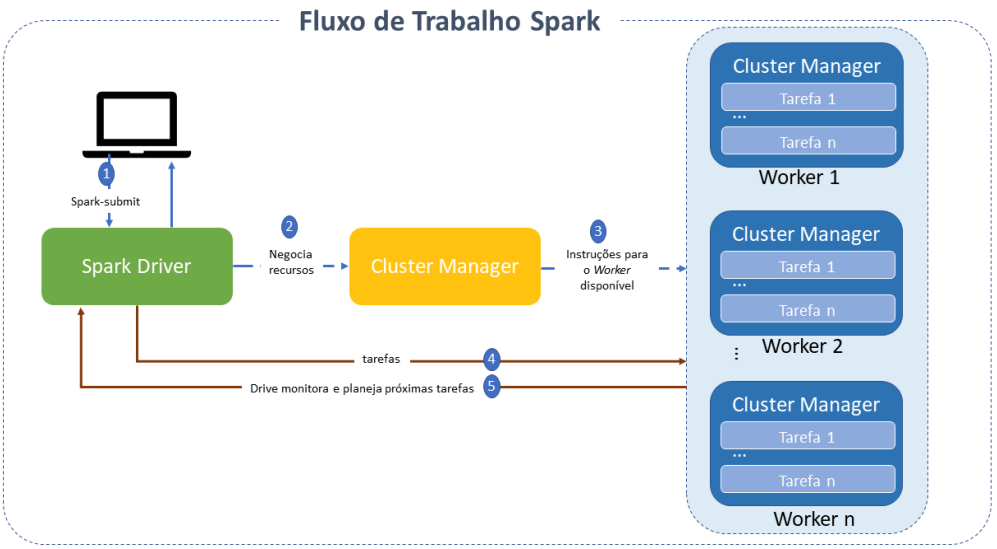
    - **Accumulator (Acumuladoras):** São variáveis compartilhadas apenas adicionadas por meio de uma operação associativa e comutativa, usadas para executar contadores (similar aos contadores MapReduce) ou operações de soma.

Fluxo de Trabalho do Apache Spark

© Copyright Tecnisys 2026. All rights reserved.

Seu ciclo de vida envolve vários passos intermediários, cada um responsável por conduzir responsabilidades específicas.

- O processo se inicia com a submissão do job pelo cliente, por meio da opção *spark-submit*.
- A classe *main*, especificada durante a submissão do job, é chamada e o programa Spark driver é iniciado no nó master, responsável por gerenciar o ciclo de vida do aplicativo.
- O programa driver solicita recursos do cluster manager para iniciar os executores com base na configuração do aplicativo.
- O cluster manager ativa o executor no nó worker em nome do driver Spark, que agora assume a propriedade do ciclo de vida do aplicativo.
- O driver Spark cria uma DAG com base no RDD. A tarefa é então dividida em etapas. O driver Spark envia as tarefas para o executor, que as executa.
- O executor envia uma solicitação de conclusão da tarefa ao driver por meio do cluster manager. Depois que todas as tarefas são concluídas em todos os executores, o driver envia um status de conclusão ao cluster manager.



Fluxo Spark

Outras Características do Apache Spark

- O Spark pode acessar **fontes de dados variáveis** e executar em **várias plataformas**, incluindo o Hadoop.
- Fornece **APIs funcionais de alto nível** em Java, Scala, Python e R para:
  - **manipulação de dados em grande escala**

© Copyright Tecnisys 2026. All rights reserved.



    - **armazenamento em cache de dados in-memory**
    - **reutilização de datasets**
- Suporta **vários formatos e conjuntos de APIs** para lidar com qualquer tipo de dado em modo distribuído.
- Oferece um **mecanismo otimizado com suporte a gráficos de execução geral**.
- Utiliza o conceito de **grafo acíclico direcionado (DAG)**, por meio do qual é possível desenvolver pipelines compostos por várias etapas complexas.
- A capacidade de armazenamento de dados *in-memory* e processamento *near real-time* torna o Spark **mais rápido que o framework MapReduce** e oferece uma **vantagem para casos de uso iterativos** onde o mesmo *dataset* é usado várias vezes em diferentes execuções.

Boas Práticas para Apache Spark

- **Uso de Dataframe/Dataset sobre RDD**: O RDD serializa e desserializa sempre que distribui os dados entre clusters. Essas operações são muito caras. Por outro lado, o Dataframe armazena os dados como binários, usando armazenamento off-heap, sem a necessidade de serialização e desserialização de dados na distribuição para clusters, tornando-se uma grande vantagem em relação ao RDD.

- **Uso do Coalesce para Reduzir o Número de Partições**: Sempre que for necessário reduzir o número de partições, use o coalesce, pois ele faz o movimento mínimo de dados sobre a partição. Por outro lado, o repartition recria toda a partição, tornando a movimentação de dados muito alta. Para aumentar o número de partições, temos que usar o repartition.

- **Uso de Formatos de Dados Serializados**: Geralmente, seja um job de streaming ou em lote, o Spark grava os resultados calculados em algum arquivo de saída, e outro job do Spark o consome, faz alguns cálculos e grava novamente em algum arquivo de saída. Nesse cenário, usar um formato de arquivo serializado, como Parquet, nos dá uma vantagem significativa sobre os formatos CSV e JSON.

- **Evitando Funções Definidas pelo Usuário**: Use as funções pré-construídas do Spark sempre que possível. As funções UDF (User Defined Functions) são uma caixa preta para o Spark, impedindo-o de aplicar otimizações. Dessa forma, perdemos todos os recursos de otimização oferecidos pelo Dataframe/Dataset do Spark.

- **Dados de Memória em Cache**: Sempre que fazemos uma sequência de transformações do Dataframe e precisamos usar um Dataframe intermediário

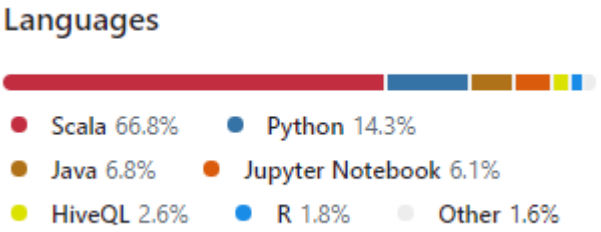
© Copyright Tecnisys 2026. All rights reserved.

repetidamente para cálculos adicionais, o Spark fornece um recurso para armazenar um DF específico na memória na forma de um cache.

Detalhes do Projeto Apache Spark

O Spark é escrito na linguagem Scala e executado em uma máquina virtual Java. Atualmente, suporta as seguintes linguagens para desenvolvimento de aplicativos:

- Scala
- Java
- Python
- Clojure
- R



Linguagens do Spark

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **4.0.2**, distribuída pelo Helm Chart `tdp-spark` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- Apache.org
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Sqoop

Transferência de Dados

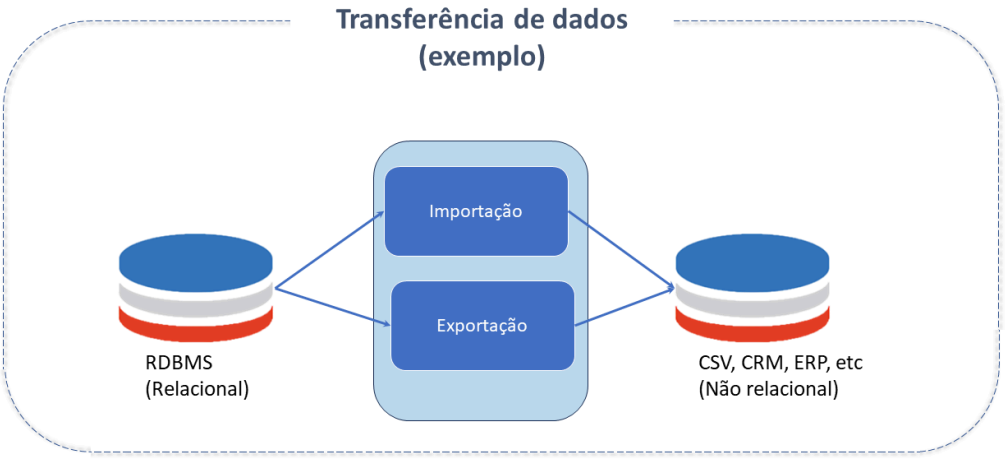
Este componente foi aposentado pela Apache Software Foundation e não recebe mais atualizações ou suporte.

Este componente será removido a partir do TDP 3.1.0.

Antes de serem processados por um sistema de aprendizado de máquina, os dados precisam ser importados. Da mesma forma, precisam ser exportados para outras aplicações antes de usados externamente.

Um "engine" de transferência de dados habilita a movimentação "para" ou "de" dispositivos de armazenamento diferentes. O trabalho real de importação é feito pelos mecanismos de processamento, que executam os trabalhos de importação e, em seguida, persistem os dados importados no dispositivo de armazenamento.

Diferentemente de outros sistemas de processamento de dados, onde os dados de entrada estão em conformidade com um esquema e são quase sempre estruturados, os dados de origem de um sistema de "machine learning" pode incluir uma mistura de fontes e formatos.



Transferência de Dados

Características do Apache Sqoop

© Copyright Tecnisys 2026. All rights reserved.



O Apache Sqoop é uma**ferramenta de linha de comando** criada para viabilizar a **transferência de dados "em massa"** entre Apache Hadoop e *datastores* estruturados, como SGBD Relacionais.

A integração destes ambientes é o papel do Sqoop e sua principal finalidade é realizar a **importação/exportação de dados** entre o ambiente relacional e o Hadoop.

O dado armazenado em *databases* externos não pode ser acessado diretamente pelas aplicações MapReduce. Esta abordagem colocaria o sistema em *nós* dos *Clusters* em grande risco de estresse.

O Sqoop simplifica o carregamento de grandes quantidades de dados de RDBMS para o Hadoop, solucionando essa questão.

A maior parte do procedimento é automatizada pelo Sqoop, que depende do banco de dados para especificar a estrutura de importação de dados. Ele importa e exporta dados usando a arquitetura MapReduce, que oferece uma abordagem paralela e tolerante a falhas.

O Sqoop graduou-se da incubadora em março de 2012, tornando-se um projeto de nível superior na Apache.

Algumas de suas características merecem destaque:

- Faz a **leitura linha por linha** da tabela ao escrever o arquivo no HDFS.
- Realiza **importação de dados e metadados** de bancos relacionais **direto para o Hive**.
- Fornece um **processamento paralelo e tolerante a falhas** ao usar o Map Reduce em atividades import/export.
- A ferramenta Sqoop **usa a estrutura YARN para importar e exportar dados**, tornando-se tolerante a falhas com paralelismo.
- Possibilidade de selecionar o **intervalo de colunas a serem importadas**.
- Possibilidade de especificar os **delimitadores e formatos de arquivos**.
- **Paraleliza conexões em Banco de Dados** executando comando SQL como SELECT(import) e Insert/Update(export)
- O **Padrão do arquivo importado do HDFS é o CSV**.
- **Conversão de tipos de dados**: O Sqoop **importa tabelas individuais ou *Databases* inteiros** para arquivos em HDFS.

© Copyright Tecnisys 2026. All rights reserved.

- Um **conector JDBC genérico** é fornecido para conexão com qualquer banco de dados que suporte o padrão JDBC. Possui diversos *plugins* para conexão com PostgreSQL, Oracle, Teradata, Netezza, Vertica, DB2, SQL Server e MySQL.
- Cria classes java que permitem a **interação de usuários com o dado importado**.

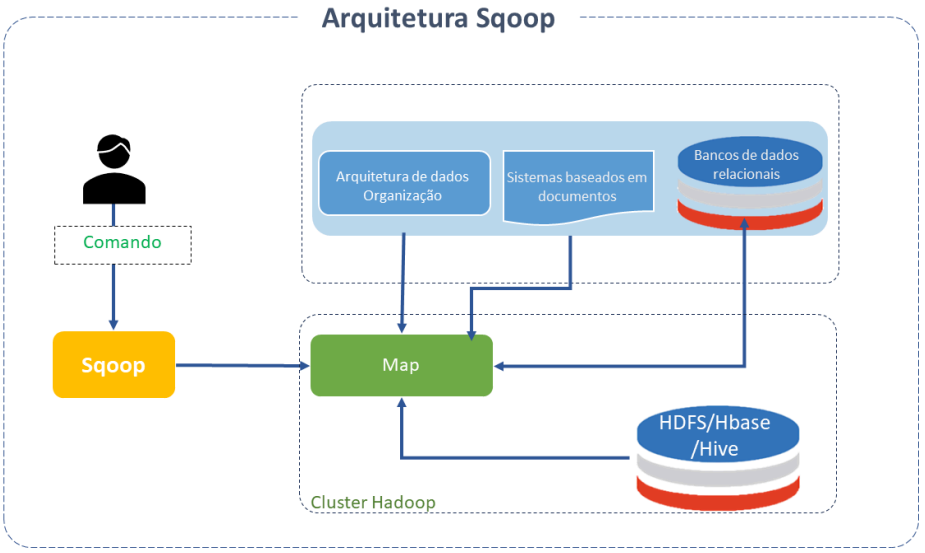
Arquitetura do Apache Sqoop

Sqoop fornece uma interface de linha de comando para usuários finais e também pode ser acessado pela API Java.

A migração de dados entre Sqoop Hadoop e um sistema externo de armazenamento é possível por meio dos **conectores Sqoop**, os quais o habilitam a usar vários bancos de dados relacionais conhecidos como, por exemplo, o MySQL, PostgreSQL, Oracle, etc.

Cada uma destas conexões pode se comunicar com o DBMS ao qual está vinculada.

O que ocorre durante a execução do Sqoop é bem simples: O *dataset* transferido é dividido em várias porções e um job somente de Map é criado com mappers distintos encarregados de carregar cada partição. O Sqoop usa as informações do banco de dados para deduzir os tipos de dados, manipulando cada registro de maneira segura.



Arquitetura Sqoop

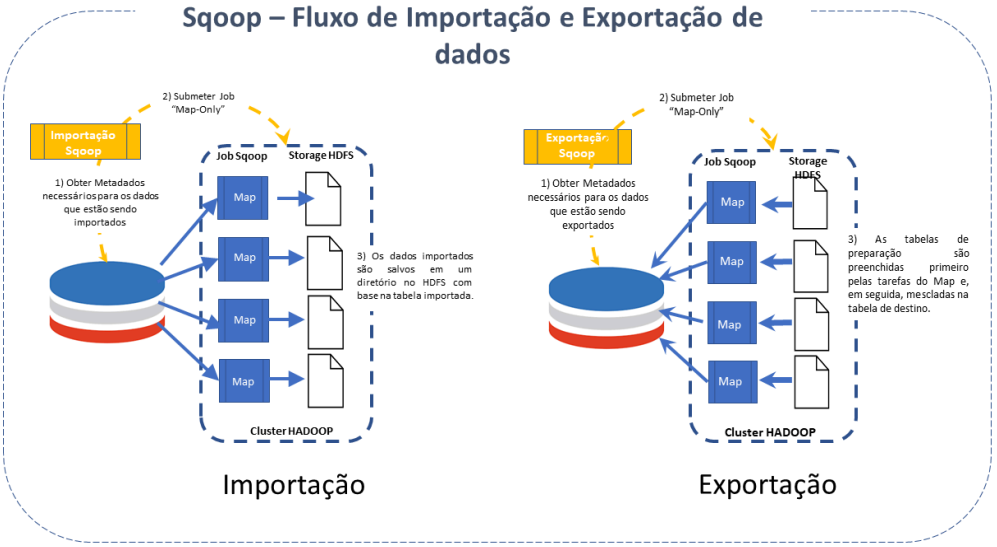
O Sqoop é composto, assim, de duas operações principais:

- Importação Sqoop

© Copyright Tecnisys 2026. All rights reserved.

Procedimento realizado com o comando *Sqoop import*. Cada registro carregado no SGBD Hadoop como um único registro é mantido em arquivos texto como parte da estrutura Hadoop. Ao importar dados, também é possível carregar e dividir o Hive. O Sqoop também permite a importação incremental de dados.

- Exportação Sqoop
  Facilita a execução da tarefa com auxilio do comando *export*, que realiza a operação no sentido contrário. Os dados são transferidos do sistema de arquivos Hadoop para o SGBD Relacional. Antes de finalizar a operação, os dados exportados são transformados em registros.



Fluxo Sqoop

- Recursos do Apache Sqoop
  - É possível importar os **resultados de uma query SQL em HDFS com Sqoop**.
  - Oferece **conectores para a maior parte dos RDBMS**, como MySQL, Microsoft SQL Server, PostgreSQL, etc.
  - **Suporta protocolo de autenticação de rede Kerberos**, permitindo que os nós autentiquem usuários enquanto se comunicam com segurança numa rede insegura.
  - Com um único comando, o Sqoop **pode recarregar a tabela inteira ou seções específicas** da tabela.

Quando usar o Apache Sqoop

- Processamento de bases OLTP usando alguma ferramenta de Big Data.
- Integração de OLTP com Hadoop

© Copyright Tecnisys 2026. All rights reserved.

- Ingestão de dados no Hadoop

Como funciona

O Comando inserido pelo usuário é analisado pelo Sqoop e executa o Hadoop *Map* apenas para importar ou exportar dados, pois a fase *Reduce* só é necessária quando as agregações são necessárias.

Sqoop analisa os argumentos inseridos na linha de comando e prepara a tarefa *Map*. Um trabalho de mapeamento que executa vários mapeadores depende do número definido pelo usuário na linha de comando.

Durante uma importação, cada tarefa do map recebe uma parte dos dados a serem importados com base na linha de comando.

Sqoop distribui os dados uniformemente entre os mapeadores para garantir alto desempenho.

Em seguida cada mapeador cria uma conexão com o banco de dados JDBC.

Boas Práticas para Apache Sqoop

- **Importação para formato binário** Embora as importações para o formato de arquivos CSV sejam fáceis de testar, alguns problemas podem surgir quando o texto armazenado no Banco de dados usa caracteres especiais. **A importação para o formato binário, como o Avro evitará este problema e poderá tornar mais rápido o processamento no Hadoop**.

- **Controlar o paralelismo** Sqoop trabalha no modelo de programação MapReduce. Importa e exporta dados a partir da maior parte de databases relacionais em paralelo. O número de tarefas *Map* por job determina este paralelismo. **Controlar o paralelismo permite lidar com a carga dos databases e também com sua performance**. Existem duas formas de explorar o paralelismo no Sqoop:

    - **Alterando o número de mappers** Os trabalhos típicos do Sqoop iniciam quatro mappers por padrão. Para otimizar o desempenho, recomenda-se aumentar as tarefas do Map (processos paralelos) para um valor inteiro de 8 ou 16. Isto pode mostrar um aumento no desempenho em alguns bancos de dados. Usando *-m* ou *--num--mappers* pode-se definir o grau de paralelismo no Sqoop.

© Copyright Tecnisys 2026. All rights reserved.



Terminal input

```
Sqoop import
--connect jdbc:postgresql://postgresql.example.com/sqoop
--username sqoop
--password sqoop
--table nome-da-tabela
--num-mappers 10
```

- **Dividindo por consulta** Ao realizar importações paralelas, o Sqoop precisa de um critério para dividir a carga de trabalho. Ele usa uma coluna de divisão para dividir a carga de trabalho.

  Por padrão, identificará a coluna de chave primária (se estiver presente) em uma tabela e a usará como coluna de divisão.

  Os valores baixo e alto para a coluna de divisão são recuperados do Banco de Dados e as tarefas de map operam em componentes de tamanho uniforme do intervalo total.

  O parâmetro split-by divide os dados da coluna uniformemente com base no número de mappers especificados. A sintaxe split by é:

Terminal input

```
Copiar código
Sqoop import
--connect jdbc:postgresql://postgresql.example.com/nome-do-database
\--username nome-do-usuário
\--password senha
\--table nome-da-tabela
\--split-by campo_id
```

> **WARNING**
>
> O número de tarefas map deve ser menor que o número máximo de conexões de databases paralelos possíveis. O incremento no grau de

© Copyright Tecnisys 2026. All rights reserved.

> paralelismo deve ser menor que aquele disponível dentro de seu Cluster Map Reduce.

- **Controle do Processo de transferência de dados**: Um método popular de melhorar a performance é gerenciando o caminho onde importamos e exportamos dados. Resumimos abaixo alguns caminhos. Para ver os argumentos existentes, Pesquise as Tabelas "Table 3" e "Table 29" -> argumentos de controle de importação/exportação, na Guia do Usuário:

  - **Batch:** (lote) Que significa que as instruções SQL podem ser agrupadas em um lote quando os dados são exportados.

    > **NOTE**
    >
    > A interface JDBC disponibiliza uma API para fazer lotes em uma instrução preparada com vários conjuntos de valores. Essa API está presente em todos os drivers JDBC porque é exigida pela interface JDBC.

    O lote é desabilitado por padrão no Sqoop. Habilite o lote JDBC usando o parâmetro *batch*.

    Terminal input

    ```
    sqoop export
    -- connect jdbc: postgresql://postgresql.example.com/_nome-do-database_
    \-- username _nome-do-usuário_
    \-- password _senha_
    \-- table _nome_tabela_
    \-- export-dir /data/_nome-do-database_
    \-- *batch*
    ```

  - **Tamanho de busca**: O número padrão de registros que podem ser importados de uma única vez é 1.000. Isso pode ser alterado pelo parâmetro *Fetch-size*, usado para especificar o número de registros que o Sqoop pode importar por vez.

© Copyright Tecnisys 2026. All rights reserved.



Terminal input

```
-- connect jdbc: postgresql://postgresql.example.com/_nome-do-database_
\-- username _nome-do-usuário_
\-- password _senha_
\-- table _nome_tabela_
\-- fetch-size=n
    */} onde n representa o número de entradas que o Sqoop deve buscar
por vez.
```

Com base na memória e largura de banda disponíveis, o valor do parâmetro *fetch-size* pode ser aumentado em relação ao volume de dados que precisa ser lido.

- Modo Direto:

Por padrão, o processo de importação Sqoop usa JDBC, que fornece um suporte razoável. No entanto, alguns bancos de dados podem obter maior desempenho usando utilitários específicos de bancos de dados pois são otimizados para fornecer a melhor velocidade de transferência possível, colocando menos pressão sobre o servidor de banco de dados .

Ao fornecer o argumento *--direct*, o Sqoop é forçado a tentar usar o canal de importação direta. Este canal pode ter um desempenho maior que usar o JDBC.

Terminal input

```
Sqoop import
-- connect jdbc: postgresql://postgresql.example.com/_nome-do-database_
\-- username _nome-do-usuário_
\-- password _senha_
\-- table _nome_tabela_
\-- direct
```

WARNING

Existem várias limitações que companham essa importação mais rápida. Nem todos os bancos de dados possuem utilitários nativos

© Copyright Tecnisys 2026. All rights reserved.

> disponiveis e esse modo não está disponível para todos os bancos de dados.

Sqoop tem suporte direto para MySQL e PostgreSQL.

Consultas de limite personalizados: Como já visto, o *split by* distribui uniformemente os dados para importação. Se a coluna tiver valores não uniformes, a consulta de limite pode ser usada se não obtivermos os resultados desejados ao usar apenas o arqumento *split-by*.

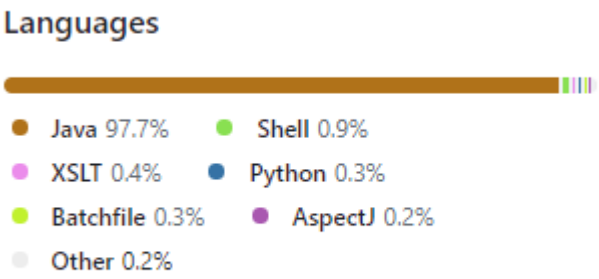
Idealmente, configuramos o parâmetro de consulta de limite como min(id) e max(id) juntamente com o nome da tabela.

Terminal input

```
Sqoop import
-- connect jdbc: postgresql://postgresql.example.com/_nome-do-database_
\-- username _nome-do-usuário_
\-- password _senha_
\-- query 'SELECT... FROM... JOIN ... USING ... WHERE $CONDITIONS'
\-- split-by id
\-- target-dir _nome-da-tabela_
\-- boundary-query "selecionar min(id), max(id) de _nome-da-tabela-
normalizada_"
```

Detalhes do Projeto Apache Sqoop

Apache Sqoop foi desenvolvido em JAVA.



Linguagens do Sqoop

© Copyright Tecnisys 2026. All rights reserved.

Notas Técnicas

> **COMPONENTE APOSENTADO — REMOÇÃO PREVISTA NO TDP 3.1.0**
>
> O Apache Sqoop foi **aposentado pela Apache Software Foundation** e movido para o Apache Attic, repositório oficial de projetos inativos. Isso significa que o projeto não recebe novas versões, correções de segurança ou suporte da comunidade.
>
> A última versão estável disponível é a **1.4.7** (lançada em 2017).
>
> **Impacto para usuários do TDP:**
>
> - O Sqoop permanece funcional nas versões atuais da plataforma, mas não é recomendado para novos projetos.
> - **A partir do TDP 3.1.0, o componente será removido da plataforma.**

Fontes:

- Apache Sqoop
- Apache Attic — Sqoop

© Copyright Tecnisys 2026. All rights reserved.

Apache Solr

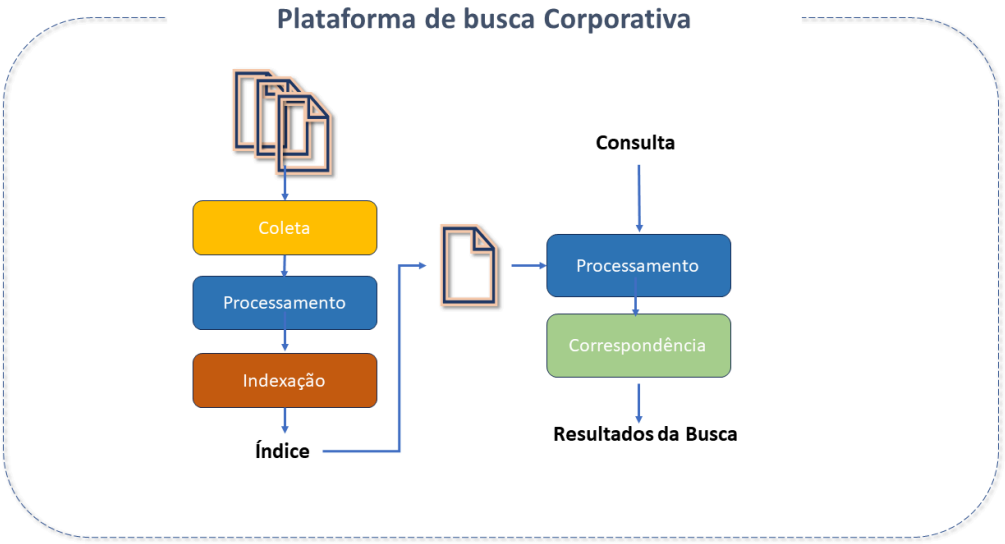
Plataforma de Busca Corporativa

Plataformas de Busca Corporativa têm a função de criar uma função de pesquisa segura, poderosa e fácil de usar.

Quando aplicadas a contextos de negócios, geralmente se integram a soluções da inteligência de negócios e gerenciamento de dados, e são usadas para "limpar" e estruturar dados, tornando as informações mais fáceis de localizar, podendo extrair informações de fontes distintas, como CRM, ERP, etc.

Para se qualificar como uma plataforma de pesquisa corporativa, é necessário que o produto:

- Trate informações de distintas fontes, tipos e formatos de dados.
- Indexe ou arquive dados.
- Forneça opções de pesquisa inteligente.
- Ofereça uma interface para pesquisa e recuperação de dados.
- Permita que usuários refinem suas pesquisas por meio de filtros avançados.
- Defina permissões para acesso a informações.



© Copyright Tecnisys 2026. All rights reserved.

Características do Apache Solr

O Apache Solr é uma plataforma de pesquisa corporativa de código aberto, construída sobre o Apache Lucene, projetada para recuperação de documentos.

O Apache Solr foi criado em 2004 por Yonik Seeley, na CNET Networks, como um projeto interno para adicionar capacidade de pesquisa ao site da empresa.

- Em janeiro de 2006 seu código fonte foi doado à Apache Software Foundation e, em 2007, já era considerado um projeto autônomo de nível superior(TLP), tendo crescido continuamente com recursos acumulados, atraindo usuários, colaboradores e *commiters*.
- Foi lançado pela Apache em 2008, na versão 1.3 e, em 2010, fundiu-se com o Lucene(seu esquema de versão foi alterado em 2011 para corresponder com o do Lucene).
- Em 2020 a *Bloomberg* doou o operador Solr para o projeto Lucene/Solr, o que viabilizou a implantação e execução do Solr no Kubernetes.
- Em 2021 o Solr foi estabelecido como um projeto Apache separado(TLP), independente do Lucene. Sua primeira versão independente foi a 9.0.

O Solr é amplamente usado para casos de uso de pesquisa e análise corporativa. É desenvolvido de forma aberta e colaborativa pelo Projeto Apache Solr, na Apache Software Foundation. Possui uma comunidade de desenvolvimento muito ativa com lançamentos regulares.

Dentre suas principais características, destacamos:

- **Recursos de busca *Advanced Full-Text*:** Habilitando recursos avançados de busca, incluindo correspondência de frases, uso de *wildcards* para flexibilizar as pesquisas, além de suportar junções e agrupamentos em qualquer tipo de dado.

- **Otimizado para tráfico de alto volume:** Tendo sido comprovado em escalas extremamente grandes em todo o mundo.

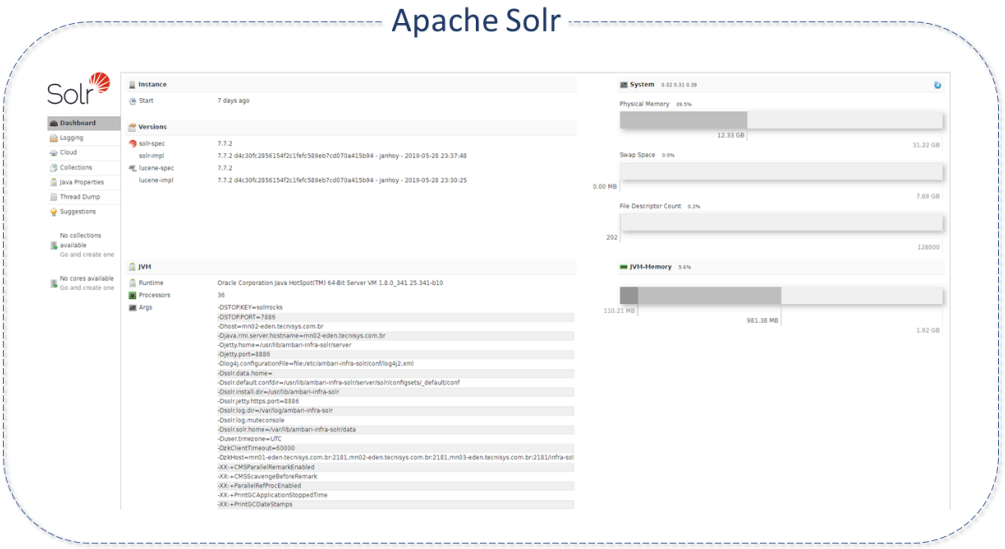
- **Interfaces abertas:** Baseadas em padrões XML, JSON e HTTP.

- **Interfaces de Administração abrangentes:** Contando com uma interface de usuário administrativa responsiva integrada para facilitar o controle de suas

© Copyright Tecnisys 2026. All rights reserved.

instâncias.

- **Fácil monitoramento:** Publicando cargas de dados métricos via JMX.
- **Altamente escalável e tolerante a falhas:**
  - Construído no Apache Zookeeper, torna fácil escalar para mais ou para menos.
  - Oferece replicação, distribuição, rebalanceamento e tolerância a falhas.
- **Flexível e adaptável com configuração simples:** Configurações de esquema flexíveis, que permitem o armazenamento de quase qualquer tipo de dado no Solr.
- **Indexação *near Real-time*:** Permitindo que alterações sejam vistas em qualquer tempo.
- **Arquitetura de Plugin extensível:** Publica muitos pontos de extensão que facilitam a incorporação de *plugins* de índice e *query time*, permitindo(por ser uma licença open-source), que seu código seja alterado à vontade.



Interface Apache Solr

Arquitetura do Apache Solr

O Apache Solr foi projetado para escalabilidade e tolerância a falhas. É executado como um servidor de pesquisa de texto autônomo.

Apache Solr executa sobre a biblioteca de software Lucene que é, de fato, a engine que o potencializa. O núcleo Solr é uma **instância do índice Lucene**, onde ficam todos os arquivos de configuração necessários para usá-lo. A Biblioteca de pesquisa Lucene Java

© Copyright Tecnisys 2026. All rights reserved.



é utilizada para indexação e pesquisa de texto completo e possui APIs HTTP/XML e JSON semelhantes a REST que tornam Solr **utilizável nas linguagens de programação mais populares**.

Sua configuração externa permite que se adapte a muitos tipos de aplicativos sem codificação Java e sua arquitetura de plug-in oferece suporte a personalizações mais avançadas.

As **consultas SOLR são URLs de solicitação HTTP simples** e as **respostas são um documento estruturado no formato JSON, XML, CSV ou outro**. Os documentos (*indexing*) são colocados nele via JSON, XML, CSV ou binário sobre HTTP. É consultado via HTTP GET e recebe resultados JSON, XML, CSV ou binário.

Solr executa numa **arquitetura não-escrava**, pois **todo nó é seu próprio master**. Os nós usam Zookeeper para aprender sobre o estado do *Cluster* e cada nó (JVM) pode hospedar vários núcleos (onde o Lucene é executado).

As consultas são configuradas e gerenciadas nos arquivos *schema.xml* e *solrconfig.xml*.

Os principais componentes da arquitetura Apache Solr são:

- RequestHandler: As requisições recebidas no Apache Solr são processadas pelo *Request Handler*. A requisição pode ser de consulta ou atualização de índice. Baseada na requisição, o *request handler* é selecionado. Geralmente, há o mapeamento do manipulador para um *end-point* URI por meio do qual a requisição será atendida.

- Recurso de Busca: Provê o recurso de busca ao Apache Solr. Podendo ser um *spell checking*, consulta, *faceting*, *hit highlighting*, etc.

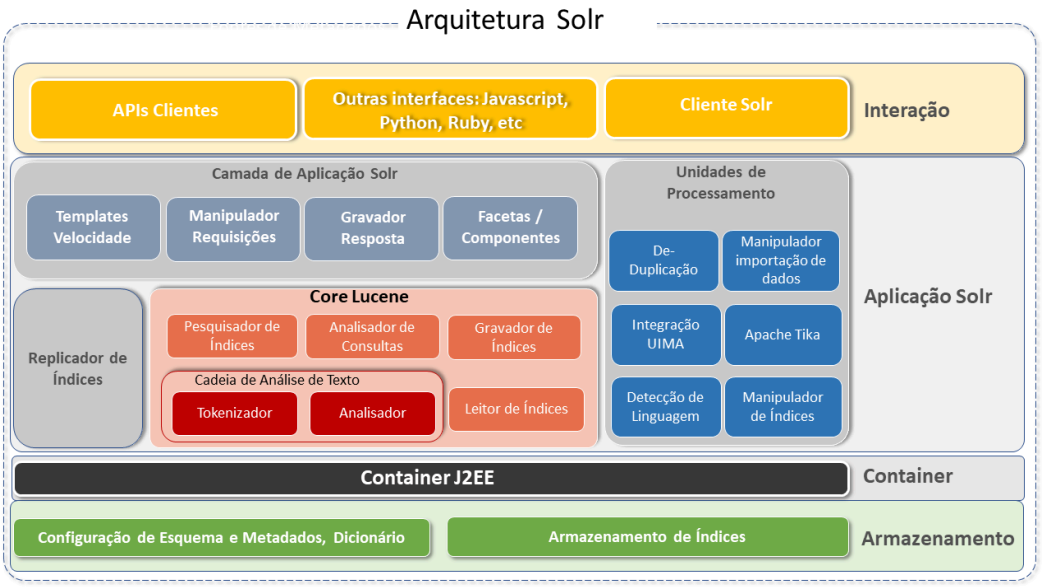
- Query Parsers (Analisadores de Consulta). Realizam a análise das consultas e verificam se existem erros de sintaxe. Após a análise, traduz as consultas em um formato compreendido pelo Lucene. O Solr suporta vários query parsers.

- Response Writer(Gravador de resposta): É o componente que gera a saída formatada para a consulta do usuário. Os formatos suportados são XML, JSON, CSV, etc. Existem diferentes *response writers* para cada tipo de respostas.

© Copyright Tecnisys 2026. All rights reserved.

- Analisador / tokenizador: Lucene reconhece dado na forma de *tokens*. O Apache Solr analiza o conteúdo, divide-o em *tokens* e repassa estes *tokens* para o Lucene. Um Analisador no Apache Solr examina o texto de campos e gera um "fluxo de token". Um tokenizador divide o *fluxo de token* preparado para o analizador em *tokens*.

- Update Request Processors (Processadores de Requisições de Update): Toda requisição de *update* (atualização) executará por meio de um conjunto de *plugins* (assinatura, registro, indexação), coletivamente conhecidos como "processador de requisição de atualização". Este processador é responsável por modificações como "remoção de campo, adição de campo, etc."



Arquitetura Apache Solr

Terminologia do Apache Solr

- **Cluster:** Nós Solr operando entre si, de forma coordenada, por meio do Zookeeper, e gerenciados como uma unidade. Um *Cluster* pode conter várias coleções.

- **Coleções:** Um ou mais documentos agrupados em um índice lógico único usando uma única configuração e Schema. Na nuvem, a coleção pode ser dividida em múltiplos fragmentos lógicos, que, por sua vez, podem ser distribuídos ao longo de vários nós ou em um único nó.

- **Nó:** Na nuvem Solr, cada instância única do Solr é considerada um nó.

- **Commit:** Viabiliza as mudanças de forma permanente no índice.

© Copyright Tecnisys 2026. All rights reserved.



- **Core:** Uma instância individual do Solr (que representa um índice lógico). Vários cores podem executar em um único nó.

- Shard(fragmento): É uma partição lógica da coleção que possui um subconjunto de documentos dela, de forma que cada documento em uma coleção esteja contido exatamente em um Shard.
  Quando um documento é enviado a um nó para indexação, o sistema primeiro determina a qual "shard" o documento pertence, e, em seguida, qual nó hospeda seu Líder, encaminhando o documento para este líder para indexação, que, por sua vez encaminhará para todas as outras réplicas.

- Replicação: Quando o índice é muito grande para uma única máquina e há um volume de consultas que "shards" únicos não possam acompanhar, é o momento de replicar cada "shard". No núcleo Solr, a réplica é uma cópia do *shard* que é executada em um nó.

- **Líder:** É a réplica do *shard* que distribui as requisições do *Solr Cloud* para as demais réplicas.

Recursos do Apache Solr

O Solr oferece recursos ricos e flexíveis para pesquisa. Seus principais recursos incluem:

- **Pesquisa de texto:** Empoderado pelo Lucene, o Solr permite recursos de correspondência como frases, *wildcards*, *joins*, agrupamentos e outros em qualquer tipo de dado.

- Highlighting (Realce de ocorrências): Permite que fragmentos de documentos que correspondam à consulta sejam incluídos na sua resposta.

- Faceting (Pesquisa facetada): É a organização dos resultados da pesquisa em categorias com base em termos indexados.

- Indexing (Indexação em tempo real):: O Solr permite que seja criado um índice com muitos campos distintos ou tipos de entrada. Um índice Solr pode aceitar dados de muitas fontes diferentes, como XML, CSV ou dados extraídos de tabelas em bancos de dados e arquivos em formatos comuns como Word ou PDF.

- Result Collapsing and Expanding (Recolhimento e Expansão de Resultados): Agrupa documentos (recolhendo o conjunto de resultados) segundo parâmetros e provê

© Copyright Tecnisys 2026. All rights reserved.



acesso ao documento no agrupamento "recolhido" para uso em "display" ou aplicativo.
- SpellChecking: Projetado para fornecer sugestões de consulta em linha com base em outros termos similares.

- **Integração com banco de dados**

- **Recursos NoSQL:** O recurso *realtime get unique-key* permite a recuperação da versão mais recente de quaisquer documentos sem o custo de reabrir um pesquisador. Isto é útil principalmente ao usar o Solr como um armazenamento de dados NoSQL e não apenas um índice de pesquisa.

  O Real Time Get depende do recurso de *log* de atualização, que é habilitado por padrão e pode ser configurado em `solrconfig.xml`.

Alguns dos sites mais famosos da Internet usam Solr, como o Macy, Ebay, Zappo.

Boas Práticas para o Apache Solr

- **Campos indexados:** O uso de armazenamento em nuvem pode ser bastante otimizado por meio do controle do número de campos indexados.
  Índices representam um fator crítico entre qualidade, performance e custo e, em grande quantidade, podem degradar performance e incrementar custos.
  Na definição de campos, procure não marcar campos como *indexed=true* se eles não são usados numa consulta.
  O número de campos indexados aumenta:

  - O uso de memória durante a indexação.
  - O tempo de mesclagem de segmentos.
  - O tempo de otimização.
  - O tamanho do índice.
- **Armazenamento de campos:** Recuperar campos armazenados do resultado de consulta pode ser oneroso. Esse custo é afetado pelo número de bytes armazenados por documento. Quanto maior a contagem de bytes, mais esparsa será a distribuição dos documentos no disco - o que demandará mais largura de banda de E/S para recuperar os campos desejados.

  Esses custos aumentam nas implementações em nuvem. Obviamente, seu uso deve avaliar qualidade, desempenho e custo.

© Copyright Tecnisys 2026. All rights reserved.



- Utilização do Cache Solr: Solr armazena em cache vários tipos de informações para garantir que consultas semelhantes não se repitam desnecessariamente.
  O Document Cache ajuda a melhorar o tempo para atender solicitações.
  Existem três tipos principais de cache:

  - *Query cache*: armazena IDs de documentos retornados pelas consultas.
  - *Filter cache*: armazena filtros criados pelo Solr em resposta aos filtros adicionados às consultas.
  - *Document cache*: armazena os campos do documento solicitado quando os resultados das consultas são mostrados.
- Autowarming: A ativação do *autowarming* pode incrementar de forma significativa a performance de uma pesquisa para qualquer dos três tipos de cache.
  Ao incrementar o *autowarmCount*, o Solr irá pré-preencher ou realizar *autowarm* no cache com os objetos de cache criados pelo resultado da busca.
  Esta configuração indica o número de itens em *cache* que serão copiados para o novo "buscador".

- Expressões Streaming: O Solr oferece uma linguagem de processamento de *streams* simples, mas poderosa para o Solr Cloud.
  Expressões de *streaming* são um conjunto de funções que podem ser combinadas para executar muitas tarefas de computação paralela.

- Heap da Memória JVM: O *heap* é uma área da memória em que são armazenados os objetos da aplicação, criados a partir de classes (com semântica de referência). Nele são alocados objetos referenciados em algum lugar.
  Esta área é gerenciada pelo *garbage collector* e vai ocupando espaços conforme a necessidade. O GC, em algum momento futuro, vai liberar os espaços quando os dados de uma determinada porção não são mais necessários.
  No caso da memória JVM (Java Virtual Machine), boa parte da alocação já é reservada com antecedência pelo GC, que tenta administrar a memória da melhor forma possível.
  As configurações de *heap* da memória JVM afetam diretamente a utilização de recursos do sistema.
  Em um ambiente de "nuvem" o uso de recursos e custos podem aumentar rapidamente, o que pode afetar diretamente a performance e o sucesso ou falha da implementação de uma busca.

© Copyright Tecnisys 2026. All rights reserved.

- **Adicionando espaço suficiente:** Várias ferramentas, geralmente incluídas nas instalações padrão Java mostram a quantidade mínima de memória consumida pela JVM ao executar o Solr (inclusive *jconsole*).
  Sem memória suficiente para o *heap*, a JVM pode aumentar o consumo de recursos devido à execução frequente do processo de *garbage collection* (coleta de lixo).
  É recomendável um adicional de 25% a 50% sobre a memória mínima necessária para operar com desempenho máximo.

- **Memória S.O.:**
  É necessário cuidado na alocação de *heap de memória JVM*, pois Solr usa extensivamente o componente *org.apache.lucene.store.MMapDirectory*.
  Este componente aproveita o Sistema Operacional(S.O.) que é usado pelo componente *MMapDirectory*.
  Desta forma, quanto mais *heap de Memória JVM* for alocada, menos memória estará disponível para o Sistema operacional, o que também prejudica o desempenho da pesquisa.

É necessário encontrar um ponto de equilíbrio.

- **Tamanho ideal do _heap de memória JVM:**
  Um intervalo entre 8 e 16 Gb é adequado. Recomenda-se iniciar com o menor valor e executar testes, monitorando o uso da memória, e ajustando, gradualmente, o tamanho, até alcançar o tamanho ideal.

- ***Garbage Collector*(Coleta de lixo):** É um termo usado para o processo em que a memória liberada em um programa Java retorna ao "pool" de memória para reutilização.
  O coletor utilizado antes da versão 9 do Java era o ParallelGCf. O Coletor G1GC opera de forma multiencadeada simultânea a aborda problemas de latência existentes no ParallelGC.
  Recomenda-se, portanto, seu uso.
  Além disso, o script de controle do Solr vem com um conjunto de configurações de coleta de lixo Java pré-configuradas que funcionam bem para várias cargas de trabalho distintas. Mas estas configurações podem não funcionar tão bem para algum uso específico.
  Neste caso, as configurações do GC podem ser alteradas, o que também deve ser feito com a variável GC_TUNE no arquivo /etc/default/solr.in.sh.

© Copyright Tecnisys 2026. All rights reserved.

Ajustes finos recomendados na configuração de produção

- Configurações de memória e GC
  Por default, o *script bin/solr* configura o tamanho máximo de heap Java para 512M (-Xmx512m), o que é um número razoável para começar a trabalhar com Solr.
  Para a produção, é adequado ajustar o tamanho máximo baseado nos requisitos de memória dos aplicativos de busca. Valores entre 10 e 20 Gb são comuns. Para mudar estas configurações, use a variável *SOLR_JAVA_MEM*:

  Terminal input

  ```
  solr_java_mem="-Xms10g -Xmx10g"
  ```

- Shutdown_ por falta de memória: O *script* bin/solr registra *script bin/oom_solr.sh* a ser chamado pela JVM se ocorrer um erro *OutOfMemory*, que emitirá um *kill -9* para o processo Solr.
  Esse comportamento é recomendado ao executar no modo SolrCloud para que o Zookeeper seja imediatamente notificado de que um nó está apresentando um erro irrecuperável.
  É recomendável inspecionar o conteúdo do *script* /opt/solr/bin/oom_solr.sh para se familiarizar com as ações que o *script* executará se for chamado pela JVM.

- Modo SolrCloud:
  Para executar o Solr no modo SolrCloud é necessário definir a ZK_HOST no arquivo de inclusão para apontar para o conjunto Zookeeper.
  A execução do Zookeeper incorporado não é suportada em ambientes de produção. Se um conjunto Zookeeper estiver hospedado, por exemplo, em três hosts na porta cliente padrão 2181(zk1, zk2, zk3), será necessário definir:

  Terminal input

  ```
  zk_host=zk1,zk2,zk3
  ```

  Quando a variável *ZK_HOST* for definida, o Solr iniciará no modo "nuvem".

- Zookeeper:
  Ao usar uma instância do Zookeeper compartilhada por outros sistemas, é

© Copyright Tecnisys 2026. All rights reserved.

recomendável isolar a "árvore" *SolrCloud znode* usando o suporte *chroot* do Zookeeper.
Para garantir que todos os *znodes* criados pelo *SolrCloud* sejam armazenados em /solr, por exemplo, use:

Terminal input

```
zk_host=zk1,zk2,zk3/solr
```

Antes de usar um *chroot* pela primeira vez, é necessário criar o caminho raiz (znode) no Zookeeper usando o script de controle Solr. O Comando *mkroot* pode ser usado:

Terminal input

```
bin/solr zk mkroot /solr -z <ZK_node>:<ZK_PORT>
```

- Hostname Solr:
  Use a variável SOLR_HOST no arquivo de inclusão para definir o nome do host do servidor Solr.

  Terminal input

  ```
  solr_host=solr1.example.com
  ```

  Isto é especialmente recomendável ao executar no modo SolrCloud, pois determina o endereço do nó quando ele se registra no Zookeeper.

- Substituição de configurações em solrconfig.xml:
  Solr permite que as propriedades de configuração sejam substituídas usando propriedades do sistema Java passadas na inicialização por meio da sintaxe *-Dproperty=value*.

- Executando vários nós Solr por host: O *bin/solr* é capaz de executar várias instâncias em uma máquina, mas para uma instalação típica, essa configuração não é recomendada.
  Recursos extras de CPU e memória serão necessários para cada instância adicional.

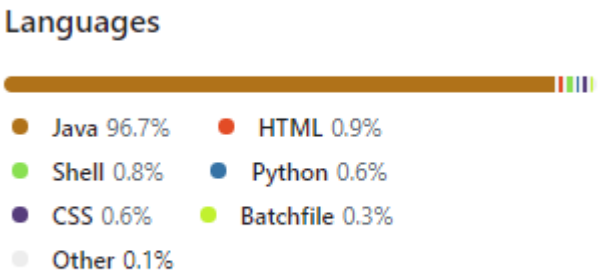
© Copyright Tecnisys 2026. All rights reserved.



> **WARNING**
>
> Esta regra não se aplica quando se discute escalabilidade extrema. A execução de vários nós Solr em um host é uma boa alternativa quando queremos reduzir a necessidade de heaps extremamente grandes. Para mais detalhes veja aqui.

Detalhes do Projeto Apache Solr

O Solr foi escrito em Java.



Linguagens do Solr

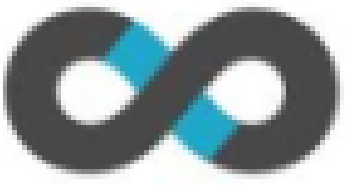
Fontes

- whismy Apache Solr
- Apache Wiki
- Apache.Solr.org
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Superset

Visualização



Visualização de dados consiste na representação gráfica de informações e dados. Elementos visuais como gráficos e mapas facilitam a narração de histórias sobre dados específicos, tornando-os mais compreensíveis, destacando tendências, exceções e gerando novos insights.

Com o advento do "Big Data", tornou-se uma ferramenta incrivelmente relevante para interpretar e entender o volume de dados gerados diariamente.

Existem diversas ferramentas para visualização de dados, e entre elas está o Apache Superset, uma ferramenta simples, fácil de usar, que oferece uma gama de opções para todos os níveis de habilidade.

É uma das melhores ferramentas de exploração de dados e aprendizado de máquina. Além disso, oferece uma interface de usuário muito amigável a um custo mais acessível.

Características do Apache Superset

Construído sobre tecnologias populares de código aberto, como JDBC e H2O, o Apache Superset oferece recursos robustos para a visualização, exploração e análise de dados. É um aplicativo web de inteligência de negócios (BI) que é rápido, leve, intuitivo e repleto de opções que facilitam a exploração e visualização de dados por usuários com qualquer conjunto de habilidades.

Foi criado por Maxime Beauchemin, Engenheiro de dados, CEO e fundador da PRESET, que usou um *hackathon* interno do Airbnb(evento que reúne programadores, designers e outros profissionais ligados ao desenvolvimento de software para uma maratona de programação) para criar uma ferramenta de BI a partir do zero.

O projeto, denominado "Caravel" inicialmente, tornou-se o Apache Superset. Rapidamente adotado por dezenas de empresas, assumiu cada vez mais casos de uso. Foi estabelecido como projeto de código aberto completo e incubado com a Apache

© Copyright Tecnisys 2026. All rights reserved.



Software Foundation em 2016. Hoje, é a principal plataforma analítica de código aberto, com uma das comunidades de crescimento mais rápido do GitHub.

Dentre suas principais características podemos citar:

- **Interface Intuitiva**: Permite visualizar conjuntos de dados e criar painéis interativos.
- **SQL IDE**: Facilita o preparo de dados para visualização, incluindo um rico navegador de metadados.
- **Segurança**: É uma das suas principais vantagens, pois oferece controle total sobre o acesso a dados. Permite a adição de usuários ao Banco de Dados, o fornecimento de acesso e o rastreamento de comportamentos.
- **Camada Semântica Leve**: Capacita analistas a definirem rapidamente dimensões e métricas personalizadas.
- **Suporte a Bancos de Dados SQL**: Compatível com a maioria dos bancos de dados que utilizam SQL.
- **Cache e Consultas Assíncronas**: Melhora a performance ao reduzir a carga de consulta diretamente nos bancos de dados.
- **Extensibilidade**: Com controle total de acesso ao dado, permitindo a configuração de regras complexas sobre quem pode acessar quais recursos e conjuntos de dados do produto.
- **Integração com os principais *backends* de autenticação:** Como OpenID, LDAP, OAuth, Remote_user, etc.
- **Capacidade de adicionar *plug-ins* de visualização personalizados**.
- **API para customização programática**.
- **Arquitetura Cloud Native**: Projetado para alta disponibilidade e escalabilidade em ambientes distribuídos.
    - Desenhada para _Alta Disponibilidade e *Escala* para ambientes grandes e distribuídos.

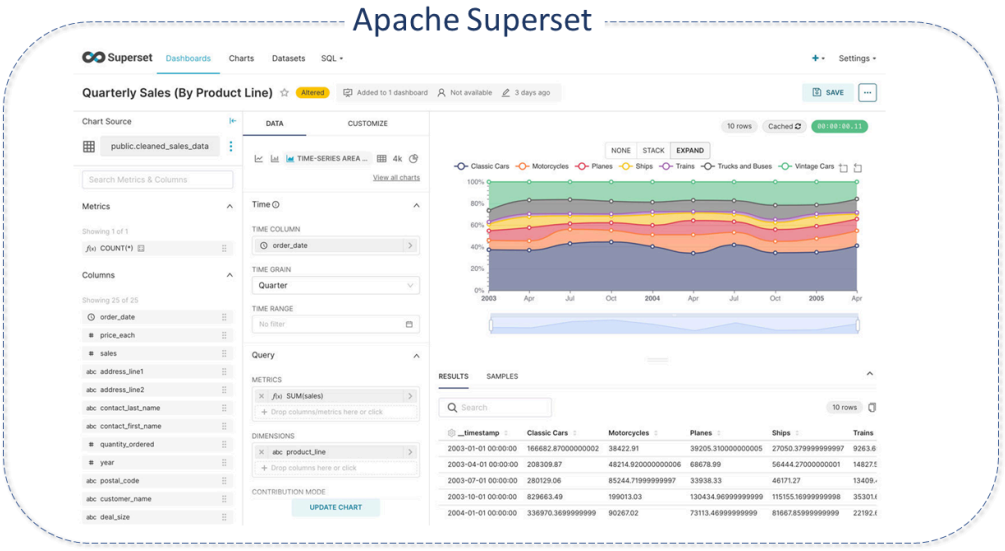
© Copyright Tecnisys 2026. All rights reserved.

    - Flexível na escolha de:
        - Web Server (Gunicorn, Nginx, Apache),
        - Database para Metadados (MySQL, PostGres, MariaDB, etc), ** Fila de mensagens (Redis, RabbitMQ, SQS, etc),
        - *Backend* de Resultados (S3, Redis, Memcached, etc),
        - Camada de *Cache* (Memcached, Redis, etc.)
- **Trabalha muito bem dentro de *Containers***.

- **Permite a criação de Queries Interativas:** Com seleção de *database*, tabelas e *schema*.

- **Não exige conhecimento de código:** É desenhado para pessoas que não conhecem código, como analistas de negócio e financeiros.

- **Acessível por Web e aplicativo:** Que operam de modo independente.



Interface do Apache Superset

Arquitetura do Apache Superset

O Apache Superset baseia-se na metodologia *Dataset-Centric* (arquitetura centrada em consultas e em semântica). Esta arquitetura promove o uso de conjuntos de dados semelhantes a um *Dataframe*, ou seja, uma estrutura tabular enriquecida que contém um subconjunto de características semânticas.

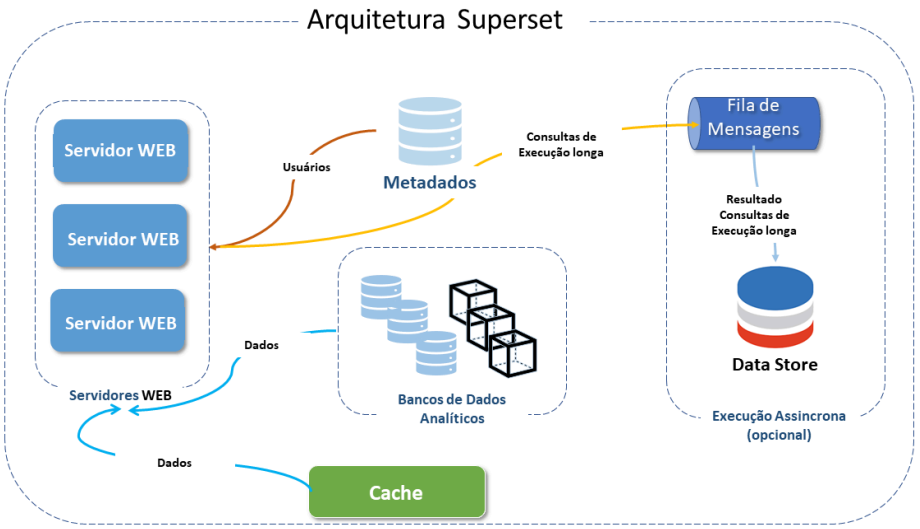
O Apache Superset pode ser executado no modo sequencial ou distribuído. No modo sequencial, apenas executa consultas com duração inferior a 60 segundos. No modo

© Copyright Tecnisys 2026. All rights reserved.

distribuído, o superset distribui as consultas entre *workers*.

Os principais componentes envolvidos na solução do Apache Superset são:

- **Web Servers:** (Servidores Web): Aplicativo flask python usado para conectar-se a qualquer banco de dados. O Superset permite a escolha do Web Server e integra-se com várias opções, como Gunicorn, NGINX, Apache HTTP. Os web servers Superset e os workers Superset Celery (opcional) podem expandir-se para quantos servidores quanto forem necessários.
- **Metadata Database**: (Database de Metadados): O Superset permite a escolha do mecanismo de banco de dados de metadados e integra-se com várias opções, como MySQL, Postgres, MariaDB, etc)
- **Cache Layer:** (Camada Cache): O Superset permite a escolha da camada de cache e integra-se com várias opções como Memcached, Redis, etc.
- **Message Queue for async queries:** (Fila de mensagens para consultas assíncronas): O Superset permite a escolha da fila de mensagens e integra-se com várias opções como S3, Redis, Memcached, etc.
- **Results Backend**: Para armazenamento e recuperação de resultados das consultas.
- ***Dashboard* e *Slices*** O Dashboard é uma interface do usuário que permite a visualização de vários gráficos e dados. Cada seção dentro do *Dashboard* é chamada *Slice*, que, por sua vez, podem estar em diversos formatos: texto, gráfico, ou, por exemplo, uma função.
- **SQL Lab** SQL Lab é um IDE SQL com uma ampla gama de recursos, com o qual é possível converter dados em gráficos, por exemplo.



Arquitetura do Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

Superset funciona muito bem com serviços de métricas e estatísticas como NewRelic , StatsD, DataDog e tem capacidade de executar cargas de trabalho analíticas nas tecnologias de Banco de Dados mais populares.

Atualmente, é executado em escala em muitas empresas, como, por exemplo, no ambiente de produção do Airbnb, onde, dentro de *kubernetes*, atende mais de 600 usuários ativos diários que visualizam mais de 100 mil gráficos por dia.

Os principais setores e empresas que adotam o Superset podem ser vistos aqui.

Recursos do Apache Superset

Superset é enriquecido com funcionalidades que suportam desde a criação de visualizações dinâmicas até análises complexas, oferecendo:

- Visualizações personalizadas para explorar e compreender os dados.
- Consultas SQL na Guia SQL para investigação de dados.
- Construção de visualização sem código, ou o SQL IDE para integração e análise de dados rápidas.
- Ingestão de dados leve e escalável que funciona na infraestrutura de dados existente, sem demandar uma camada de ingestão separada.
- Camada semântica básica, onde é possível controlar como as fontes de dados são exibidas e tratadas.

Integração com Databases

Superset provê funcionalidades para conexão com vários databases. Conecta-se com quase todos os principais bancos de dados, o que facilita a visualização e análise de seus dados. É compatível com Apache Spark SQL, PostgreSQL , GoogleSheets, Amazon Athena, Amazon Redshift, Azure MS SQL, etc.

Tipos de Visualização

O Apache Superset fornece uma ampla variedade de gráficos, tabelas e layouts. Alguns exemplos são:

- Gráfico de Dispersão.
- *Grids*.
- Polígonos.
- *Path*.
- *Screen Grids*.

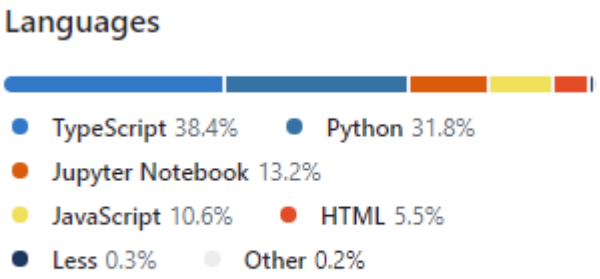
© Copyright Tecnisys 2026. All rights reserved.

- *Acrs*

Detalhes do Projeto Apache Superset

Desenvolvido predominantemente em Python, o Apache Superset também utiliza tecnologias como Typescript e Flask App Builder para sua funcionalidade interna. Suporta a versão Python 3.6 ou superior e pode ser instalado de várias maneiras, incluindo local, virtual e através de Docker.

- **localmente:** onde o Python deve ser instalado primeiro e então o *pip* instala as dependências.
- **virtualmente:** É fortemente recomendada a instalação em um ambiente virtual. O *pyenv-virtualenv* pode ser instalado se o *pyenv* estiver sendo usado.
- **Docker:** O meio mais simples de experimentar o Superset localmente é usando o Docker e o Docker Compose em Linux ou Mac OSX.



Linguagens do Superset

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **5.0.0**, distribuída pelo Helm Chart `tdp-superset` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

© Copyright Tecnisys 2026. All rights reserved.

- Comunidade Apache Superset
- GitHub Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

Apache Tez

Framework de execução de tarefas distribuídas



O Apache Tez é um framework de execução de tarefas distribuídas construído sobre o YARN (Yet Another Resource Negotiator) do Hadoop. Ele foi criado para substituir o antigo MapReduce, oferecendo um modelo de execução mais flexível, eficiente e rápido para aplicações de processamento em larga escala

O que faz o Apache Tez

- Permite a execução de DAGs (Directed Acyclic Graphs), onde cada nó representa uma tarefa (map, reduce, transformações intermediárias) e cada aresta define o fluxo de dados.
- Fornece uma camada de orquestração eficiente, reduzindo o número de leituras/escritas no HDFS e otimizando o uso de recursos no cluster.
- É a base de execução padrão do Apache Hive, acelerando consultas SQL complexas.
- Substitui processos pesados e rígidos por pipelines de dados mais curtos e otimizados.

Principais Características do Apache Tez

- **Execução baseada em DAGs**: maior flexibilidade em relação ao modelo fixo Map → Shuffle → Reduce.
- **Baixa latência**: reduz acessos desnecessários ao disco e ao HDFS.
- **Reutilização de containers YARN**: diminui overhead no agendamento e na inicialização de tarefas.
- **Suporte nativo a múltiplos padrões de comunicação** (1-para-1, broadcast, scatter-gather).

© Copyright Tecnisys 2026. All rights reserved.

- **Integração com ferramentas de alto nível** como Hive, Pig e aplicações customizadas em Java.
- **Monitoramento detalhado via WebUI e logs**
- **Suporte a falhas** herdado do YARN.

MapReduce vs. Apache Tez

O MapReduce acessa o disco/HDFS várias vezes ao longo do fluxo de dados (Mapper → Shuffle & Sort → Reducer). Ele lê e escreve dados (ou dados modificados) em cada uma dessas etapas, resultando em 5–6 acessos ao disco para um único job de MapReduce.

Já o Tez lê os dados do disco, executa todas as etapas, mantém resultados intermediários em memória, aplica vetorização (processa lotes de linhas em vez de uma por vez) e então produz a saída.

Enquanto o MapReduce realiza múltiplas leituras/gravações no HDFS, o Tez evita acessos desnecessários ao HDFS.

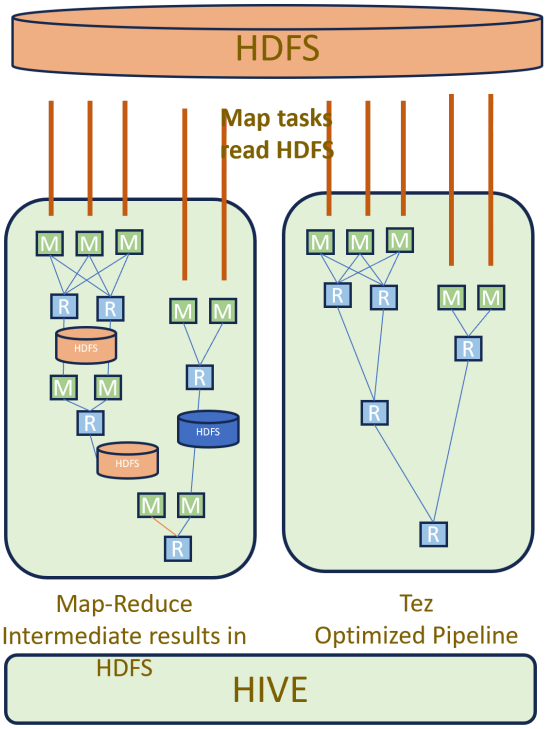


Figura 1 - Comparação entre MapReduce e Apache Tez

Arquitetura do Apache Tez

© Copyright Tecnisys 2026. All rights reserved.

A arquitetura do Tez se apoia em três pilares:

- **YARN Resource Manager**

  Responsável por gerenciar os recursos do cluster (CPU, memória).

  Aloca containers para as aplicações Tez.

- **Tez Application Master (AM)**

  Instância dedicada por aplicação DAG.

  Coordena a execução das tarefas e gerencia o pipeline.

  Mantém informações sobre progresso, falhas e dependências.

- **DAG e Tasks**

  Cada aplicação submetida ao Tez é descrita como um DAG.

  Tarefas são distribuídas em vertices (operações lógicas) e edges (conexões de dados).

  Execução é dinâmica: containers podem ser reaproveitados entre tarefas.

© Copyright Tecnisys 2026. All rights reserved.

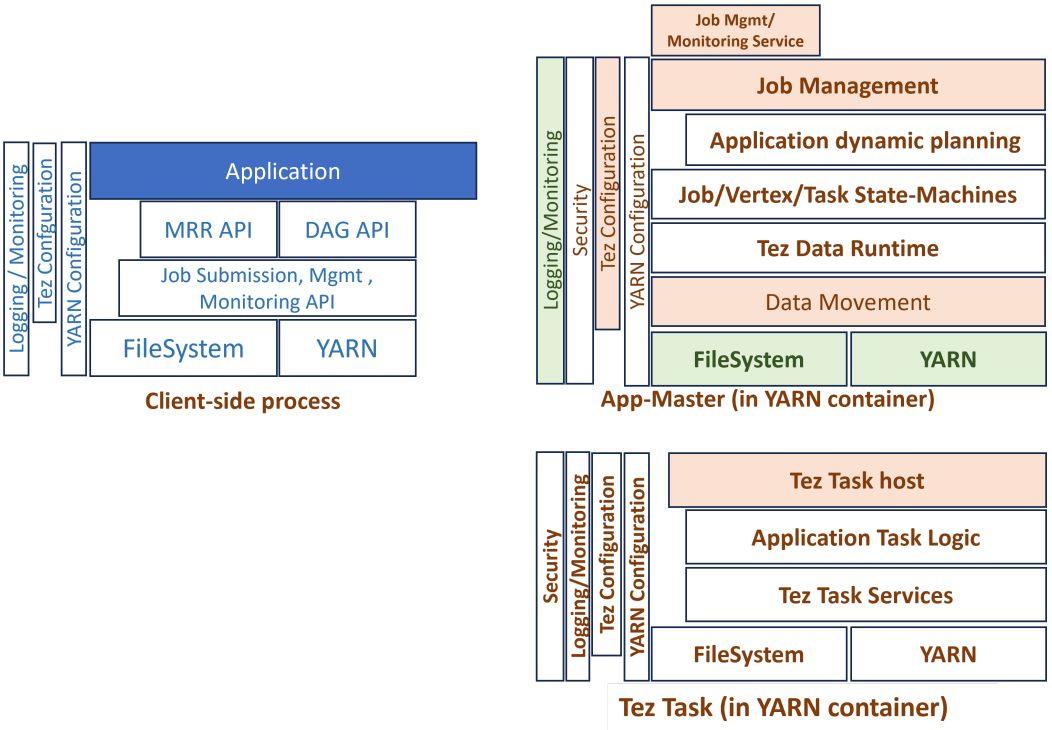


Figura 2 - Arquitetura do Apache Tez

Recursos do Apache Tez

- **APIs programáticas para criação de DAGs customizados**.
- **Extensibilidade**: desenvolvedores podem definir estratégias próprias de particionamento, ordenação e comunicação.
- **Integração com motores SQL** (Hive e Pig).
- **Tez UI**: interface para visualização de DAGs em execução, tempos de cada etapa e gargalos.
- **Configuração avançada para otimização** (memória, paralelismo, shuffle, reuso de containers).

Boas práticas

- Ajustar paralelismo de acordo com o volume de dados para evitar tanto subutilização quanto sobrecarga.
- Habilitar reuso de containers para workloads repetitivos (reduz overhead de inicialização).

© Copyright Tecnisys 2026. All rights reserved.



- Monitorar gargalos no shuffle: tuning de memória e compressão faz grande diferença.
- Usar compressão intermediária para reduzir tráfego de rede.
- Alinhar com Hive/Pig configs: muitas otimizações de Tez dependem da camada superior.

Quando usar o Apache Tez

- **Em consultas SQL complexas no Hive (substitui MapReduce)**.
- **Quando é necessário pipeline de dados com múltiplas etapas em DAGs dinâmicos**.
- **Para aplicações customizadas em Java que demandam execução paralela otimizada sobre YARN**.
- **Em cenários de Big Data interativo, onde a latência de segundos importa**.

Quando não usar o Apache Tez

- Em jobs simples de batch onde o overhead de configurar DAGs não compensa.
- Em clusters pequenos, sem YARN, ou onde MapReduce já é suficiente.
- Quando há exigência por streaming em tempo real (caso de uso típico de Kafka Streams ou Flink).

Detalhes do Projeto Tez

Java é a linguagem principal usada para implementar o Tez e suas APIs.

APIs para DAGs são escritas em Java, mas normalmente o Tez é consumido indiretamente por usuários através do Hive ou Pig (SQL ou scripts declarativos).

Fontes:

- Tez - Overview

© Copyright Tecnisys 2026. All rights reserved.

Trino

Motor de Consulta Distribuído



> **Recurso disponível somente a partir da versão 2.3.**

Um mecanismo de consulta distribuído é projetado para executar consultas SQL complexas em grandes volumes de dados espalhados por várias fontes. Elimina a necessidade de mover ou duplicar dados, permitindo análises diretamente onde os dados estão armazenados, aumentando a eficiência e a escalabilidade.

Principais características de um mecanismo de consulta distribuído:

- **Execução Paralela:** Divide consultas em tarefas menores, que são processadas simultaneamente em diferentes nós de um cluster, otimizando o tempo de execução.
- **Conexão com Múltiplas Fontes de Dados:** Oferece integração com data lakes, data warehouses e bancos de dados relacionais, permitindo a execução de consultas federadas.
- **SQL como Linguagem Universal:** Analistas podem realizar consultas utilizando SQL, sem necessidade de aprender linguagens específicas.
- **Desempenho Optimizado:** Utiliza técnicas como *pushdown* de predicados e armazenamento em memória para reduzir a latência e maximizar a performance.

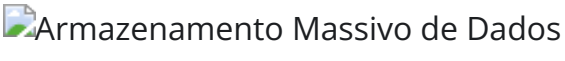
Entre os casos de uso comuns, de mecanismos de consulta distribuídos, destacam-se:

- Consultas federadas em várias fontes de dados..
- Análises em tempo real de grandes volumes de dados.
- Unificação de data lakes e data warehouses para facilitar a governança e auditoria..

O seu funcionamento básico envolve:

1. O cliente envia uma consulta ao mecanismo.
2. O mecanismo interpreta a consulta, cria um plano de execução e distribui tarefas para os *workers*.
3. Cada *worker* processa a sua parte, acedendo diretamente às fontes de dados.
4. Os resultados são consolidados e enviados de volta ao cliente.

© Copyright Tecnisys 2026. All rights reserved.



Armazenamento Massivo de Dados

Principais Características do Trino

O Trino é um mecanismo de consulta distribuído **open-source**, projetado para executar consultas SQL em grandes volumes de dados armazenados em fontes diversas, como data lakes, data warehouses e bases de dados relacionais.

Inicialmente, desenvolvido como Presto, por engenheiros do Facebook para atender suas necessidades internas de análise de dados, o Trino surgiu em 2020 após uma divisão com a Presto Foundation. Desde então, tornou-se referência em soluções de alta performance para análise de dados distribuídos.

Principais Características:

- **Velocidade:** Desenvolvido para análises de baixa latência, o Trino utiliza execução altamente paralela e distribuída para processar consultas de forma eficiente.
- **Escalabilidade Horizontal:** Permite a adição de workers para ampliar a capacidade de processamento, sendo capaz de lidar com workloads em escala de exabytes, como em grandes data lakes e data warehouses.
- **Simplicidade:** Compatível com ANSI SQL, facilita a integração com ferramentas de BI como R, Tableau, Power BI, Superset, entre outras.
- **Versatilidade:** Suporta análises "ad hoc" interativas, consultas em lote de longa duração e aplicativos de alto volume, garantindo tempos de resposta inferiores a um segundo em cenários críticos.
- **Análise Local:** Consulta dados diretamente em fontes como Hadoop, S3, Cassandra e MySQL, eliminando a necessidade de copiar ou mover dados, simplificando processos e reduzindo erros.
- **Consultas Federadas:** Habilita a execução de consultas em múltiplas fontes de dados, como HDFS, S3, bancos relacionais e data warehouses.
- **Alta Performance:** Sua arquitetura é otimizada para workloads interativos, garantindo latência mínima.
- **Extensibilidade:** Oferece suporte para conectores personalizados, possibilitando integração com novas fontes de dados.
- **Confiabilidade:** Amplamente utilizado em operações críticas, como relatórios financeiros para mercados públicos, por algumas das maiores organizações globais.
- **Suporte a Diversos Formatos:** Compatível com formatos como Parquet, ORC, Avro, JSON e CSV.

© Copyright Tecnisys 2026. All rights reserved.



- **Comunidade Aberta:** Desenvolvido sob a liderança da Trino Software Foundation, uma organização sem fins lucrativos.

Arquitetura do Trino

Trino é um mecanismo de consulta distribuído que processa dados em paralelo em vários servidores. Os servidores de um Cluster Trino são classificados como Coordinators e Workers.

As seções a seguir descrevem os principais componentes da arquitetura do Trino.

Cluster

Um cluster Trino é composto por vários nós, incluindo um Coordinator e zero ou mais Workers. Os usuários se conectam ao Coordinator por meio de ferramentas de consulta SQL. O Coordinator orquestra tarefas entre os Workers e acessa fontes de dados conectadas por meio de catálogos configurados.

Cada consulta é processada como uma operação com estado. O Coordinator distribui a carga de trabalho entre os Workers em paralelo. Cada nó executa uma única instância JVM, com paralelização adicional usando threads.

Nó

Um Nó (*Node*) no Trino refere-se a qualquer servidor dentro de um cluster que executa um processo Trino. Ele normalmente corresponde a um único computador, já que apenas um processo Trino é recomendado por máquina.

Coordinator

O Coordinator é o servidor central responsável por analisar instruções, planejar consultas e gerenciar nós Workers. Atuando como o "cérebro" do cluster, ele rastreia as atividades dos Workers, coordena a execução de consultas e se comunica com clientes e Workers por meio da API REST.

Para desenvolvimento ou testes, uma única instância do Trino pode ser configurada para funcionar como Coordinator e Worker.

Worker

Um Worker é um servidor responsável por executar tarefas e processar dados. Os Workers buscam dados de conectores, trocam dados intermediários e se comunicam com o Coordinator via API REST. Quando iniciado, um Worker registra-se no servidor de descoberta do Coordinator para alocação de tarefas.

© Copyright Tecnisys 2026. All rights reserved.

Cliente

Clientes se conectam ao Trino para enviar consultas SQL e recuperar resultados. Eles podem acessar fontes de dados configuradas por meio de catálogos e incluem ferramentas como interfaces de linha de comando, aplicativos desktop e sistemas baseados na web. Alguns clientes também suportam autoria interativa de consultas, visualizações e relatórios.

Fonte de Dados

O Trino suporta consultas em diversas fontes de dados, incluindo data lakes, bancos de dados relacionais e armazenamentos de chave-valor. O acesso a essas fontes de dados é configurado usando catálogos, que definem os conectores necessários, credenciais e outros parâmetros.

A seguir apresentamos os conceitos fudamentais associados à Fontes de Dados no Trino:

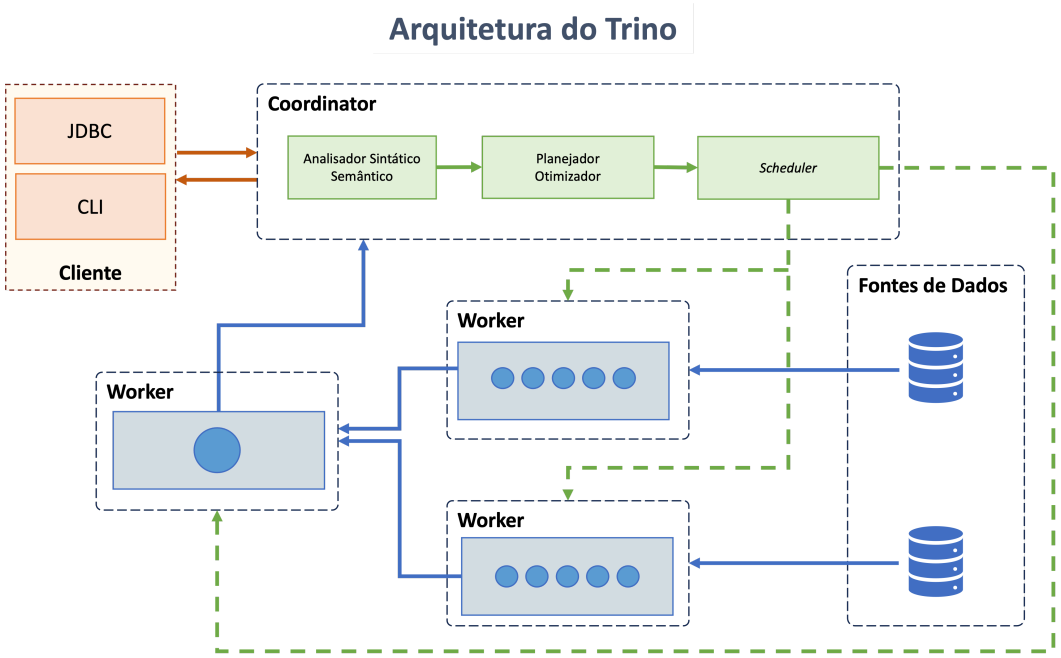
- **Conector:** Os conectores permitem que o Trino interaja com fontes de dados específicas, funcionando como drivers de banco de dados. Exemplos incluem conectores para Hive, Iceberg, PostgreSQL, MySQL e Snowflake. Cada catálogo no Trino está associado a um conector.

- **Catálogo:** Um catálogo é uma coleção de propriedades de configuração para acessar uma fonte de dados. Catálogos são definidos em arquivos de propriedades armazenados no diretório de configuração do Trino. Um catálogo pode conter esquemas e tabelas, permitindo o acesso a múltiplas fontes de dados em um único cluster.

- **Esquema:** Esquemas organizam tabelas e outros objetos dentro de um catálogo. Eles correspondem a conceitos semelhantes em bancos de dados como Hive e MySQL.

- **Tabela:** Uma tabela consiste em linhas desordenadas organizadas em colunas nomeadas com tipos específicos. As tabelas são acessadas por nomes totalmente qualificados enraizados em catálogos.

© Copyright Tecnisys 2026. All rights reserved.



Arquitetura do Trino

Modelo de Execução de Consultas do Trino

O Trino executa instruções SQL transformando-as em consultas distribuídas executadas no cluster.

Seguem mais detalhes sobre os termos utilizados no modelo de execução de consultas do Trino:

- **Instrução:** Uma instrução (*statement*) refere-se ao texto SQL enviado ao Trino. Ela é convertida em um plano de consulta durante a execução.
- **Consulta:** Uma consulta (*query*) engloba os componentes e a configuração necessários para executar uma instrução. Ela inclui estágios, tarefas, divisões e operadores.
- **Estágio:** Estágios (*stages*) representam seções de um plano de consulta distribuído, organizados hierarquicamente. Cada consulta possui um estágio raiz que agrega as saídas de outros estágios.
- **Tarefa:** Tarefas (*tasks*) executam estágios em paralelo nos Workers. Elas operam em divisões e envolvem vários drivers para processar os dados.
- **Split:** Um *split* (divisão) é um segmento de um conjunto de dados maior, processado por uma tarefa. O Coordinator atribui *splits* às tarefas com base na

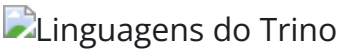
© Copyright Tecnisys 2026. All rights reserved.

disponibilidade.

- **Driver:** Drivers são as menores unidades de paralelismo, combinando operadores para processar dados dentro de uma tarefa.
- **Operador:** Operadores realizam transformações nos dados, como buscas em tabelas ou filtros, e são combinados dentro dos drivers.
- ***Exchanges*:** *Exchanges* (trocas) transferem dados entre nós durante a execução da consulta, permitindo comunicação entre estágios por meio de buffers de saída e clientes de troca.

Detalhes do Projeto Trino

O Trino foi desenvolvido em **Java**, aproveitando a robustez da JVM para processamento distribuído.



Linguagens do Trino

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão atual é **478**, distribuída pelo Helm Chart `tdp-trino` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes: Trino - Overview

© Copyright Tecnisys 2026. All rights reserved.

Apache YARN

Gerenciamento de Recursos



O Processamento de Big Data é uma tarefa difícil e não é viável em um sistema centralizado. Soluções de computação distribuídas precisam ser adotadas para viabilizar o processamento paralelo exigido pela tecnologia.

Nesse ambiente, vários locatários, com diferentes demandas, podem compartilhar recursos de computação como dados, armazenamento, rede, memória e CPU, o que torna o Gerenciamento de recursos uma tarefa crítica nesta tecnologia.

Usuários de Big Data podem solicitar vários processamentos, cada qual com distintos requisitos.

O Gerenciador de recursos age como um "scheduler", que "agenda" e prioriza solicitações, segundo os requisitos. Age, essencialmente, como um árbitro, que gerencia e aloca recursos disponíveis.

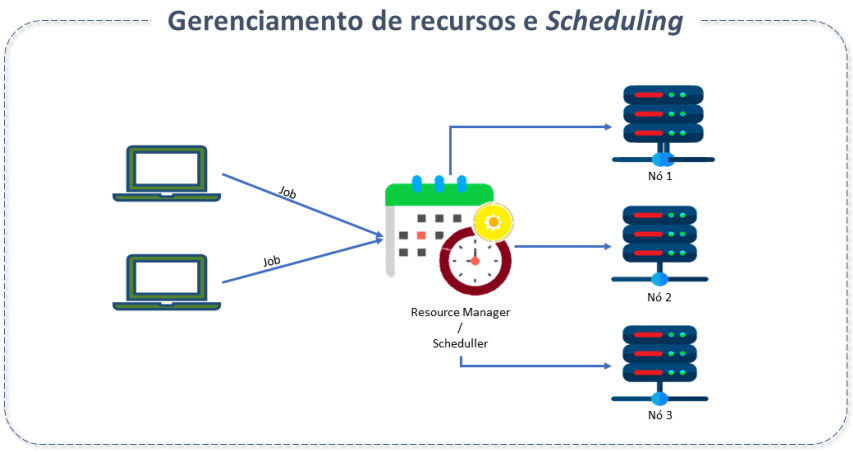
Nas primeiras versões do Hadoop, não havia um Gerenciador de Recursos. O Hadoop consistia em duas partes principais:

- **Armazenamento:** Realizado pelo HDFS
- **Processamento:** Realizado pelo MapReduce. O único job que podia ser executado e submetido ao Hadoop até a versão 2.

  As tarefas de alocação de recursos e *scheduling* eram realizadas pelo serviço *JobTracker* do MapReduce.

Com a evolução das soluções, onde processamento *real time* e *near real time* passaram a predominar, tornou-se essencial contar com um executor de aplicativos e um gerenciador de recursos para agendar e executar todos os tipos de aplicativos, incluindo MapReduce, em tempo real.

© Copyright Tecnisys 2026. All rights reserved.



Gerenciamento de Recursos e Scheduling

Características do Apache YARN

O Yet Another Resource Negotiator (YARN) foi introduzido no Apache Hadoop 2.0 para atender estas necessidades e, ainda, solucionar problemas de escalabilidade e capacidade de gerenciamento que existiam na versão anterior do Hadoop. Tem a responsabilidade de auxiliar no gerenciamento de recursos entre os *Clusters*, assumindo, ainda, a responsabilidade pelo *scheduling* e alocação de recursos para o Sistema Hadoop, tarefas realizadas até então pelo *JobTracker* do MapReduce.

Hoje, o Apache YARN ganhou popularidade devido às vantagens que oferece em escalabilidade e flexibilidade e, ainda, pela sua versatilidade e baixo custo, pois pode ser utilizado em hardwares comuns. É implementado com sucesso no eBay, Facebook, Spotify, Xing, Yahoo, etc.

Dentre suas principais características, destacamos:

- **Multilocação:** Um conjunto abrangente de limites é fornecido para evitar que um único aplicativo, usuário ou fila monopolize os recursos da fila ou do *Cluster*, evitando, assim, sobrecarga do *Cluster*.

- **Docker for YARN:** O Linux Container Executor (LCE) permite que o YARN NodeManager inicie *containers* YARN para execução diretamente na máquina *host* ou dentro de *containers* Docker.

  Os *containers* Docker fornecem um ambiente de execução personalizado no qual o código do aplicativo é executado, isolado do ambiente de execução do NodeManager e de outros aplicativos.

© Copyright Tecnisys 2026. All rights reserved.



O Docker for YARN fornece consistência (todo *container* YARN terá o mesmo ambiente de software) e isolamento (não há interferência com o que está instalado na máquina física).

- **Escalabilidade:** A escalabilidade do Apache YARN é conhecida por escalar para milhares de nós. É determinada pelo *Resource Manager* e é proporcional ao número de nós, aplicações ativas, *containers* ativos e frequência de *heartbeat* (de nós e aplicativos).

- **Alta disponibilidade para componentes:** A tolerância a falhas é um principio básico do Apache YARN. Essa responsabilidade é delegada ao Resource Manager(falhas de NodeManager e Application Master) e ApplicationMaster (falhas de *container*).

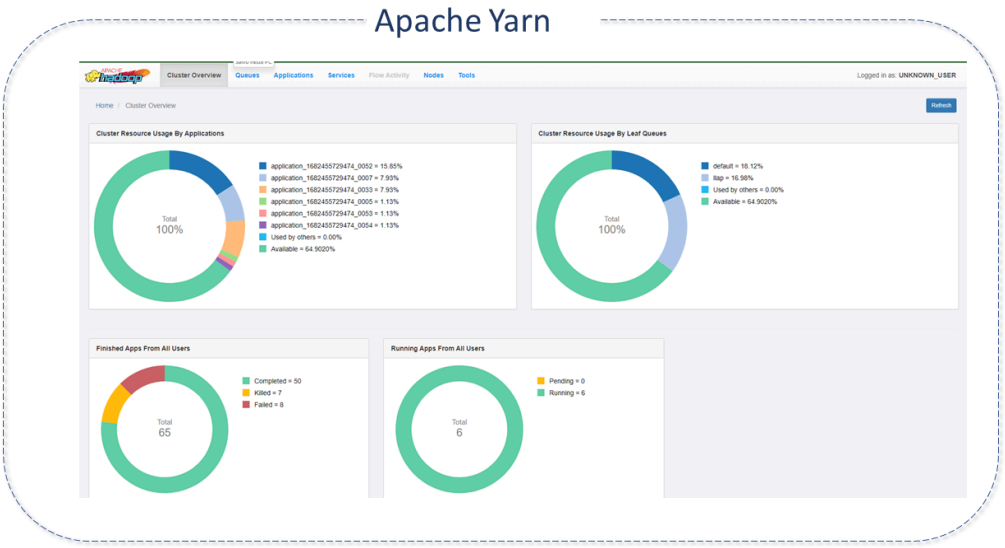
- **Modelo de Recursos flexíveis:** No Apache YARN, uma solicitação de recursos é definida em termos de memória, CPU, localidade, etc, resultando em uma definição genérica. A capacidade dos nós NodeManager e Workers é calculada baseado na memória instalada e nos núcleos da CPU.

- **Múltiplos algoritmos de processamento de dados:** o Apache YARN foi desenvolvido para executar em ampla varidade de processamento de dados. É uma estrutura para gerenciamento de recursos genéricos e permite execução de vários algoritmos sobre os dados.

- **Agregação de log e localização de recursos:** Para gerenciar *log* de usuários o Apache YARN utiliza a *agregação de logs*. Assim que o aplicativo é concluído, o serviço NodeManager agrega os *logs* de usuários relacionados a um aplicativo e grava os *logs* agregados em um único arquivo de log no HDFS.

- **Recursos eficientes e confiáveis:** o Apache YARN fornece uma estrutura genérica de gerenciamento de recursos com suporte à análise de dados por meio de vários algoritmos de processamento de dados.

© Copyright Tecnisys 2026. All rights reserved.



Interface YARN

Arquitetura do Apache YARN

o Apache YARN consiste em três componentes:

- **Resource Manager:** É o nó master, responsável pelo gerenciamento de recursos no *Cluster*. É a "autoridade maior" que arbitra recursos entre todos os aplicativos no sistema.

  Existe um Resource Manager por *Cluster* e ele "conhece" a localização e recursos de todos os *escravos*, o que inclui informações como GPU, CPU e memória necessária para execução de aplicativos. O Resource Manager age como um *proxy* entre o cliente e todos os outros nós.

  Possui dois componentes principais:

  - **Scheduler:** Responsável pela alocação de recursos para os diversos aplicativos em execução.

    O Scheduler não realiza monitoramento ou acompanhamento do *status* do aplicativo e não oferece garantias sobre reinicialização de tarefas comprometidas por falha de aplicativo ou hardware. Sua função é executada com base nos requisitos de recursos dos aplicativos, e na noção abstrata de *container* de recursos, que incorpora elementos como memória, cpu, disco, rede, etc.

© Copyright Tecnisys 2026. All rights reserved.



> **NOTE**
>
> *Containers* são parcelas de recursos (CPU, memória, etc) de uma máquina do *Cluster* que pode ser reservada para a execução de uma aplicação, sendo esta aplicação Hadoop Map Reduce ou não.

A partir da versão 2.0 do Hadoop, a coordenação de um *Job* não ocorre no nó Master, mas em um *container* instanciado em uma máquina *worker*, a qual possui geralmente dois componentes (podendo possuir três, no caso da instância de gerenciamento da aplicação):o Node Manager, Application Master e DataNode.

O Scheduler possui uma política "plugável" responsável pelo particionamento dos recursos do *Cluster* entre as várias filas, aplicativos, etc.

O Scheduler recebe solicitações do *Application Masters* com requisições de recursos e executa sua função de agendamento. A estratégia de agendamento é configurável e pode ser escolhida com base na necessidade de cada aplicativo.

Existem, por padrão, três agendadores no Apache YARN:

- **Scheduler FIFO:** Usa a estratégia simples primeiro a entrar, primeiro a "servir". A memória será alocada baseado na sequência de tempo de solicitação, sendo que o primeiro aplicativo da fila recebe a memória necessária, depois o segundo e assim por diante. Caso a memória não esteja disponível, os aplicativos aguardarão a disponibilidade. Nesta opção, o Apache YARN cria uma fila de solicitações, adicionando aplicativos a ela e os iniciando, um a um.

- Scheduler de Capacidade: Garante que o usuário obtenha a quantidade mínima de recursos. Ajuda a compartilhar os recursos do *Cluster* de maneira econômica entre diferentes usuários. Em outras palavras, os recursos do *Cluster* são compartilhados entre os vários grupos de usuários. Trabalha com o conceito de filas. Um *Cluster* é dividido em partições (filas) e cada fila recebe uma porcentagem de recursos.

© Copyright Tecnisys 2026. All rights reserved.



        - Scheduler Fair: Todos os aplicativos obtêm uma quantidade quase idêntica dos recursos disponíveis. Quando o primeiro aplicativo é enviado ao Apache YARN, este atribuirá todos os recursos disponíveis a ele. Se um novo aplicativo é enviado, o Apache YARN começa a alocar recursos para ele até que ambos tenham quase a mesma quantidade. Ao contrário dos anteriores, o Scheduler Fair evita que aplicativos fiquem sem recursos e garante que todos os aplicativos da fila obtenham a memória necessária para execução.
    - **Application Manager:** A tarefa principal do Application Manager é aceitar submissões de jobs, negociar o primeiro container para execução do Application Master específico do aplicativo e prover o serviço para reiniciar o container ApplicationMaster quando houver falha.
- NodeManager: Responsável por iniciar e monitorar *containers* de jobs. É o framework *da máquina* responsável pelos *containers*, monitorando seu uso de recursos e reportando-se ao *ResourceManager/Scheduler*.

    - **Containers:** O Hadoop 2.0 melhorou o seu processamento paralelo com a adição de containers, que são uma noção abstrata, que suporta a multilocação em um nó de dados.

      Esta foi a forma que Hadoop encontrou para definir requisitos de memória, CPU, rede: dividindo os recursos no servidor de dados em containers. Assim, o servidor de dados pode hospedar múltiplos jobs, hospedando múltiplos containers.

      O Resource Manager é responsável por "schedular" recursos alocando containers.

      Isto é feito com base em um algoritmo, a partir da "entrada" fornecida pelo cliente, a capacidade do Cluster e as filas e priorizações de recursos no *Cluster*.

      Uma regra geral é iniciar o container no mesmo nó que os dados requeridos pelo job para facilitar a sua localização.

- **Application Master:** É uma biblioteca específica do framework, encarregada de negociar recursos do *ResourceManager* e trabalhar com o(s) *NodeManager(s)* para executar e monitorar jobs. Ao receber um aplicativo, o ResourceManager iniciará o ApplicationMaster em um container alocado, que se comunicará com o *Cluster*

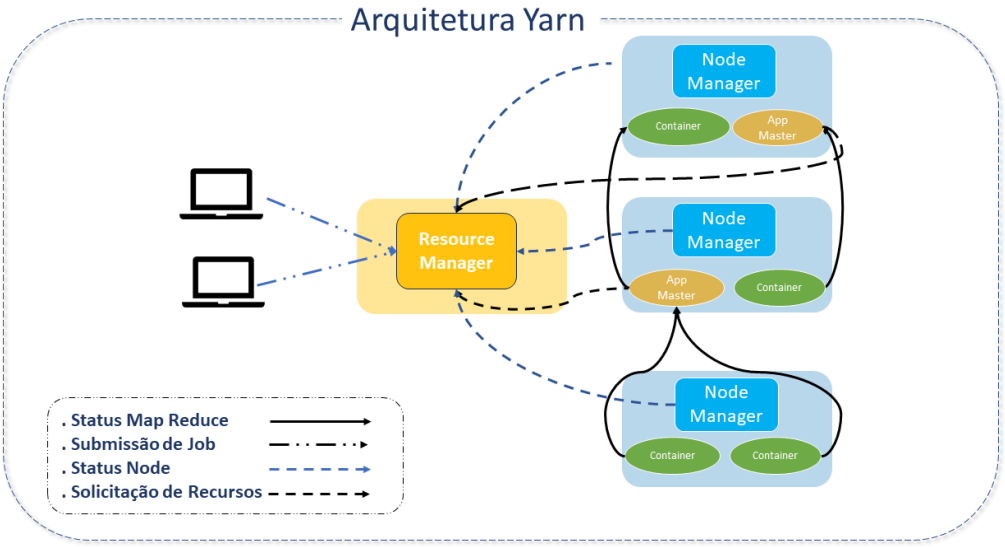
© Copyright Tecnisys 2026. All rights reserved.

YARN para lidar com a execução do aplicativo. As principais tarefas do ApplicationMaster são:

- Comunicar-se com o ResourceManager para negociar e alocar recursos para futuros containers.
- Após a alocação do container, comunicar-se com os NodeManagers para que iniciem os containers do aplicativo.

> **NOTE**
>
> O *Resource Manager* e o *Node Manager* formam o *framework* da computação de dados.

A ideia principal do Apache YARN é a de dividir as funcionalidades de gerenciamento de recursos e *scheduling*/monitoramento de jobs em *daemons* separados, existindo, assim, um *ResourceManager(RM)* global e um *ApplicationMaster(AM)* por aplicativo. Um aplicativo é um job único ou um DAG(Directed acyclic graph) de jobs.



Arquitetura YARN

> **NOTE**
>
> Com a implementação do Apache YARN na versão 2.0 do Hadoop, o MapReduce mantém a compatibilidade com a versão estável anterior (Hadoop

© Copyright Tecnisys 2026. All rights reserved.



> versão 1.x). Todas as tarefas do MapReduce continuam sendo executadas no Apache YARN.

Recursos suportados pelo Apache YARN

- Modelo de Recursos Extensível: Apache YARN oferece suporte a um modelo de recursos extensível. Por padrão, rastreia CPU e memória para todos os nós, aplicativos e filas, mas a definição de recursos pode ser estendida para incluir recursos "contabilizáveis" (recursos consumidos enquanto o *container* está em execução, mas liberados posteriormente, como memória e CPU), o que significa dizer que a definição de recursos pode ser estendida em seus valores default (como CPU e memória) para qualquer tipo de recurso que possa ser consumido quando a tarefa "executa" no *container*. Além disso, YARN oferece suporte ao uso de "perfis de recursos", permitindo que um usuário especifique várias solicitações de recursos por meio de um único perfil, semelhante aos tipos de instância Amazon Web Services Elastic Compute Cluster. É uma forma fácil de requisitar recursos a partir de um único perfil e um meio de administradores regularem o consumo.

- Reserva de Recursos: O Apache YARN oferece suporte à *reserva de recursos* por meio do *ReservationSystem*, um componente que permite ao usuário especificar um perfil de recursos ao longo do tempo e restrições temporais (por exemplo prazos) e reservar recursos para garantir a execução previsível de jobs importantes. O *ReservationSystem* rastreia recursos extras, executa o controle de admissão para reservas e instrui dinamicamente o *Scheduler* subjacente para garantir que a reserva seja atendida.

- Federation: O Apache YARN oferece suporte à *Federation* por meio do recurso *YARN Federation*, que permite conectar de forma transparente vários (sub)*Clusters* de YARN e fazê-los aparecer como um único *Cluster* massivo. Pode ser usado para alcançar uma escala maior e/ou permitir que vários *Clusters* independentes sejam usados juntos para trabalhos muito grandes ou para *inquilinos* que tenham capacidade entre todos eles. A abordagem é dividir um grande *Cluster* em unidades menores, chamadas *sub-clusters*. Cada unidade possui seu próprio *YARN Resource Manager* e *compute-node*(Node_manager).

- REST APIs: O Apache YARN oferece APIs RESTful para permitir que aplicativos cliente acessem diferentes dados de métricas como *Cluster metrics*, *schedulers*, nós, estado dos aplicativos, prioridades, gerenciadores de recursos, etc. Provê,

© Copyright Tecnisys 2026. All rights reserved.

ainda, informações e estatísticas sobre a instância NameNode e estatísticas sobre aplicativos e *containers*. Estes serviços podem ser usados por aplicativos de monitoramento remoto. Atualmente, os seguintes componentes suportam informações RESTful:

    - Resource Manager
    - Application Master
    - History Server
    - Node Manager.
- Alta Disponibilidade: Uma falha no gerenciador de recursos provocará uma falha no Apache YARN e, portanto, é importante implementar a alta disponibilidade do *Resource Manager*. O Recurso de Alta disponibilidade (HA) adiciona redundância para remover pontos de falha.

    - **Arquitetura do *Resource Manager* HA:** A alta disponibilidade do *Resource Manager* funciona na arquitetura _Active/Standby. O *Resource Manager Standby* assume o controle ao receber o sinal do ZooKeeper.

    - **Recursos do *Resource Manager* HA:**

    - **Armazenamento do *Resource Manager state*:** Em caso de falha, o *Resource Manager Standby* recarregará o estado de armazenamento e iniciará a partir do último ponto de execução. As informações do *Cluster* serão reconstruídas quando o Nodemanager enviar um *heartbeat* para o novo *Resource Manager*.

    - Resource Manager restart and failover: O *Resource Manager* carrega o estado do aplicativo interno a partir do armazenamento de estado do *Resource Manager*. O *Scheduler* do *Resource Manager* reconstrói seu estado de *cluster information* quando o NodeManager envia o *heartbeat*. O processo de *checkpoint* evita reinicialização de tarefas já concluídas.

    - **Failover and fencing:** Em *Clusters* YARN de alta disponibilidade pode haver dois ou mais *Resource Manager* em modo ativo/Standby. É possível acontecer o *split brain*, quando dois *Resource Manager* se assumem como ativos. Se isto acontecer, ambos controlarão recursos do *Clusters* e manipularão solicitações dos clientes. O *failover fencing* permite que o *Resource Manager* ativo restrinja a operação dos outros. O *armazenamento de estado* discutido anteriormente fornece o *ZKResourceManager StateStore* baseado no Zookeeper, que permite apenas um único *Resource Manager* gravando por vez.

© Copyright Tecnisys 2026. All rights reserved.

    - **Leader Elector:** Baseado no *Zookeeper ActiveStandbyElector* , é usado para eleger um novo *Resource Manager* ativo e implementar *fencing* internamente. Quando um Resource Manager fica inativo, um novo *Resource Manager* é eleito pelo ActiveStandbyElector e assumirá o controle. Se o *Failover* automático não estiver ativado, o administrador deverá realizar a transição do RM ativo para o modo de espera e vice versa manualmente.
- Nodelabels: Trata-se de um *marcador* para cada máquina, para que máquinas com o mesmo nome de etiqueta possam ser usadas em jobs específicos.

  Os nós com recursos de processamento mais poderosos podem ser rotulados com o mesmo nome e, em seguida, os jobs que requerem mais máquina podem usar o mesmo rótulo durante o envio. Cada nó pode ter apenas um rótulo atribuído a ele, o que significa que o *Cluster* terá um conjunto desarticulado de nós. Podemos dizer que um *cluster* é particionado com base nos rótulos dos nós.

  O Apache YARN também fornece recursos para definir a configuração de nível de fila, que define quanto de uma partição ou fila pode ser usado. Existem dois tipos de rótulos de nó disponíveis até o momento:

    - **Exclusivo:** garante que seja a única fila permitida para acesar o rótulo de nó. o aplicativo enviado pela fila com um rótulo exclusivo terá acesso exclusivo à partição para que nenhuma outra fila possa obter recursos.

    - **Não exclusivo:** permite o compartilhamento de recursos ociosos com outros aplicativos. As filas recebem rótulos de nó e os aplicativos enviados a essas filas terão prioridade sobre os respectivos rótulos de nó. Se não houver aplicativo ou job numa fila para esses rótulos, os recursos serão compartilhados entre outros rótulos de nó não exclusivos. Se a fila com o rótulo de nó enviar um aplicativo ou job entre o processamento, os recursos serão retirados das tarefas em execução e atribuídos às filas associadas com base na prioridade.

- Node Attributes: São uma forma de descrever atributos de um nó sem garantia de recursos. Pode ser usado por aplicações para selecionar os nós corretos para que seu *container* seja colocado com base na expressão de vários desses atributos.

- Proxy Web Application: Sua principal finalidade é reduzir a possibilidade de ataques baseados na Web por meio do Apache YARN.

© Copyright Tecnisys 2026. All rights reserved.

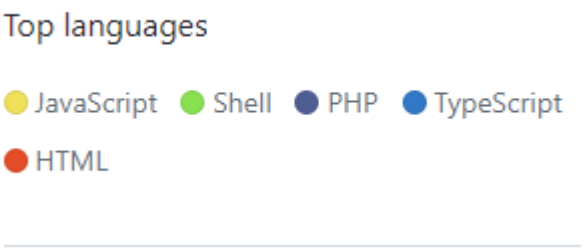
- Timeline Server e Timeline Server 2: O Timeline Server é o responsável pelo armazenamento e recuperação de informações atuais e históricas da aplicação. Possui, basicamente, duas responsabilidades:

    - Persistência de Informações específicas do aplicativo:
    - Informações genéricas persistentes sobre aplicativos concluídos.

  O Timeline Server 2 é a próxima grande iteração do Timeline Server, criada para corrigir questões de escalabilidade e melhorar a usabilidade do Timeline Server 1.

Detalhes do Projeto Apache YARN

O Apache YARN foi desenvolvido em Javascript, Shell, PHP, TypeScript e HTML.



Linguagens do YARN

Fonte(s): Hadoop.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Zeppelin

Notebook



Tipicamente utilizado por cientistas de dados em experimentos e tarefas exploratórias, o notebook é uma ferramenta de computação interativa que permite aos usuários escrever e executar código, visualizar resultados e compartilhar insights.

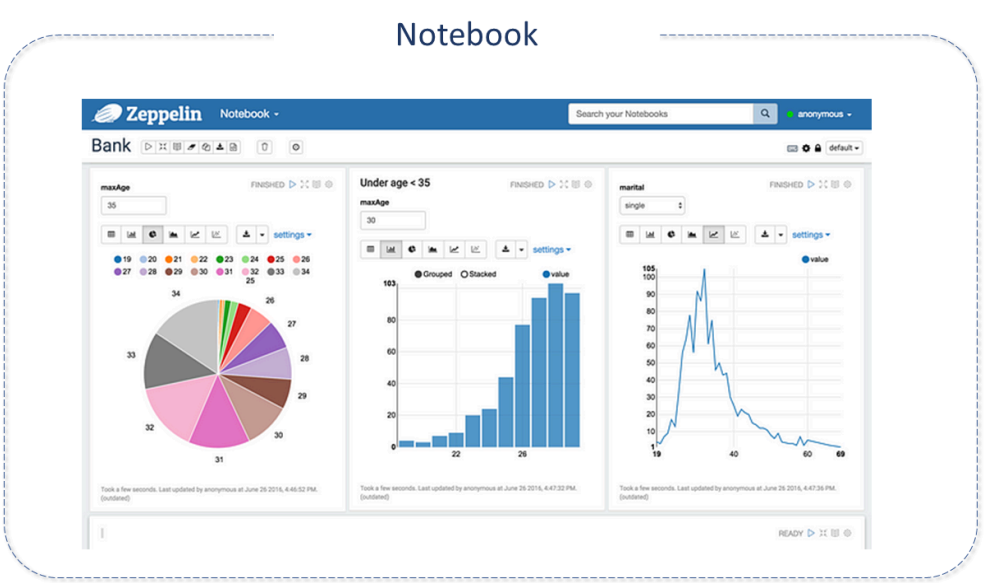
O conceito foi criado por Stephan Wolfram, cientista da computação e físico, que apresentou o Mathematica - a primeira interface computacional para notebook. Desde então, a ferramenta se proliferou e passou da "academia" para a indústria.

Apache Zeppelin é um "notebook" baseado na web que fornece recursos de ingestão, exploração, visualização, compartilhamento e colaboração interativa de dados para Hadoop e Spark.

O Apache Zeppelin começou com um projeto nomeado Zeppelin, da empresa ZEPL (anteriormente conhecida como NFLabs), comandada por Moon Sool Lee. Em 2014, tornou-se um projeto de incubadora na fundação de software Apache e logo, em 2016, se tornou um projeto de alto nível na Apache.

Segundo seus criadores a denominação de "notebook" dada ao Zeppelin é uma analogia a um caderno de anotações, baseando suas funções em "parágrafos".

© Copyright Tecnisys 2026. All rights reserved.



Notebook

Características do Apache Zeppelin

Os notebooks interativos do Zeppelin permitem a engenheiros, analistas e cientistas de dados otimizar o trabalho com dados. Desta forma, **pode ser visto como uma interface que conecta usuários com as tecnologias que desejam utilizar para tratamento de dados**.

O Zeppelin é muito útil **para trabalhar de forma interativa com workflows na ciência de dados**, desenvolvendo, organizando e executando **análises e** visualizando seus **resultados**, sem a necessidade de utilizar a *linha de comando* ou consultar detalhes do *Cluster*.

Oferece, ainda, **suporte a back-ends de vários idiomas** e um **ecossistema crescente de fonte de dados**.

- **Back-end de vários idiomas**: O software se destaca pela sua **capacidade de integrar diversas outras tecnologias através de uma funcionalidade denominada *interpreter***, que é uma camada para integração de *backend* (que trabalha "por trás" da aplicação), sendo que já possui mais de 20 interpretadores em seu pacote de distribuição oficial.

  Dentre os diversos interpretadores que suporta, podemos citar o Apache Spark, Python, JDBC, Markdown e Shell.

© Copyright Tecnisys 2026. All rights reserved.



- **Integração com Apache Spark**: O Zeppelin está integrado ao Apache Spark. Não há necessidade de criar um módulo, plug-in ou biblioteca separada para ele. Esta integração fornece:
    - SparkContext e SQLContext automáticos.
    - Carga de "dependência" JAR em tempo de execução a partir do sistema de arquivos local ou repositório "Maven".
    - Cancelamento de trabalho com exibição do progresso.
- **Visualização de Dados**: Alguns gráficos básicos também podem ser utilizados.

  As visualizações não se limitam a query SPARKSQL.

  Qualquer saída de qualquer linguagem "backend" pode ser reconhecida e visualizada.

    - **Gráficos dinâmicos**: Apache Zeppelin agrega valores e os mostra em gráficos dinâmicos. Com um simples drag-and-drop, é possível criar um gráfico com múltiplos valores agregados, incluindo soma, contagem, média, mínimos e máximos.

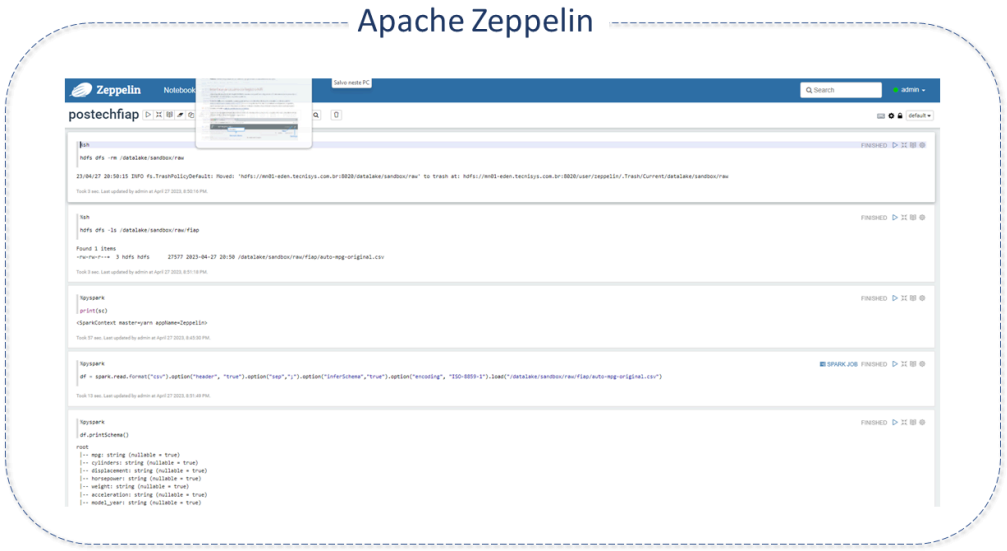
    - **Formulários dinâmicos**: O Apache Zeppelin pode criar dinamicamente alguns formulários de entrada em seu notebook.

- **Colaboração de Notebook e Parágrafos**: A URL do notebook pode ser compartilhada e, em seguida, o Zeppelin pode transmitir todas as alterações em tempo real, bem como a colaboração no Google Docs.

  Pode, ainda, fornecer uma URL para exibir apenas o resultado, numa página sem menu ou botão, dentro do Notebook, que pode ser incorporada facilmente como um iframe no site do usuário.

- **100% Open Source**: O Apache Zeppelin é licenciado Apache2. Possui uma comunidade de desenvolvimento muito ativa.

© Copyright Tecnisys 2026. All rights reserved.



Interface Zeppelin

Arquitetura do Apache Zeppelin

O Apache Zeppelin é dividido em 03 camadas:

- **Front-end**: A partir de um Web Browser, o usuario tem contato com o *Frontend* do Zeppelin, que é **baseado em *AngularJS*** (uma plataforma para construção de aplicações web baseada em *Ecmascript*) e ***Twitter Bootstrap*** (um framework *CSS*) que tornam a interface de aplicação web mais fluida e dinâmica.

  A camada de *front-end* **comunica-se com o servidor** do Apache Zeppelin **por meio de duas interfaces possíveis**:

  - **REST**: Estilo de arquitetura para definir **restrições e propriedades do protocolo *HTTP***.
  - **Web Socket**: Tecnologia para **comunicação bidirecional por canais *full-duplex*** - transmissor e receptor podem transmitir dados simultaneamente em ambos os sentidos sobre um único socket TCP.
- **Zeppelin Server**: O servidor **opera em uma Máquina Virtual** (JVM - Java Virtual Machine) que também atua como interpretador do Notebook.

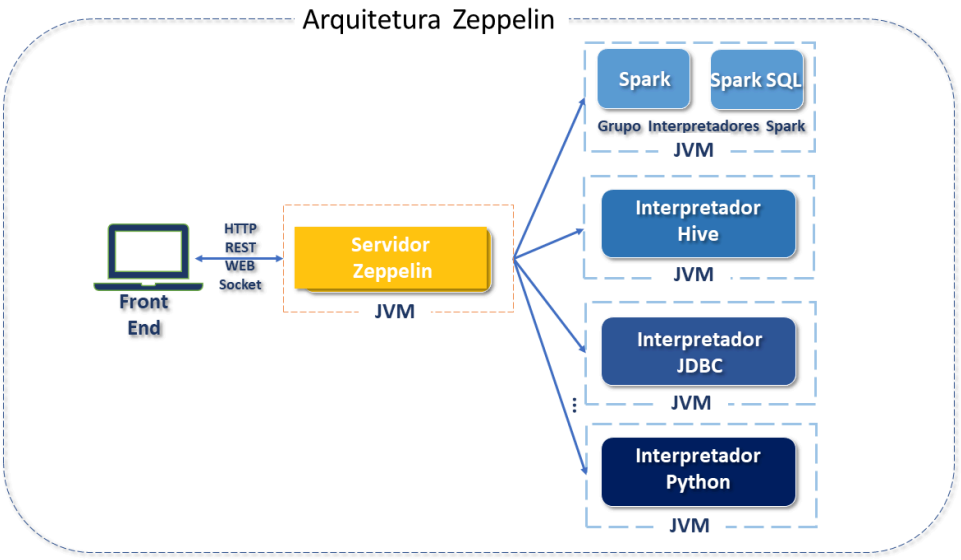
- **Interpreter**: O intérpreter **se comunica com um programa que executa em plano de fundo do Zeppelin por meio do *Apache Thrift***, uma tecnologia que permite **definir tipos de dados e interfaces de serviço** em um arquivo de definição simples.

© Copyright Tecnisys 2026. All rights reserved.

Tomando esse arquivo como entrada, o compilador gera o código a ser usado para criar facilmente clientes e servidores que se comunicam facilmente entre as linguagens de programação.

O *interpreter* é uma funcionalidade que torna o Zeppelin "plugável" a outras tecnologias. Cada processo do interpretador pertence a um grupo de interpretadores que atuam como uma unidade de start e stop do interpretador.

Para conhecer todos os interpretadores que o Apache Zeppelin suporta, clique https://zeppelin.apache.org/docs/latest/#available-interpreters.



Arquitetura Zeppelin

Visualização de dados com Zeppelin

O Notebook é composto por **parágrafos**.

Cada parágrafo é uma caixa que **recebe algum tipo de script pré-definido nos interpretadores**.

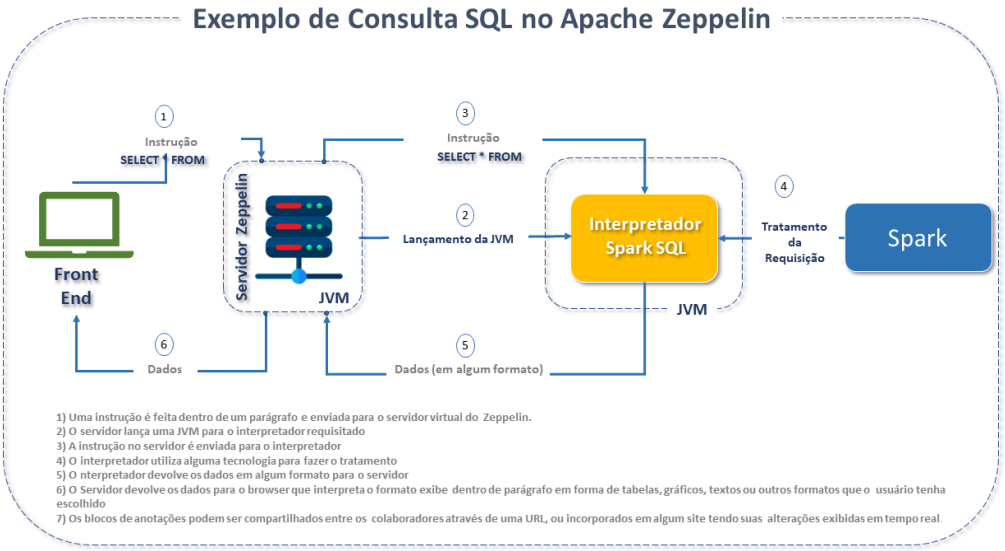
O texto interpretado possui uma marcação "%" para determinar o interpretador e script a ser executado.

Por meio da interface construída com *Angular* e *Bootstrap*, o usuário tem a possibilidade de personalizar sua visualização, colocando os parágrafos em colunas para permitir uma exibição simultânea do resultado.

© Copyright Tecnisys 2026. All rights reserved.

A interação entre o usuario, a ferramenta e os dados é proporcionada pelo front-end.

A figura abaixo é um exemplo do fluxo de funcionamento do Zeppelin, utilizando um *Backend* Spark para fazer o tratamento de uma consulta SQL.



Exemplo Zeppelin

Boas Práticas para o Apache Zeppelin

- **Instalação e Versões**: É recomendável a instalação do **Zeppelin com Ambari** e sempre uso o da **última versão** do Zeppelin, garantindo, assim, alinhamento com correções de segurança e estabilidade.

- **Escolhas de Implementação**: Embora qualquer nó possa ser selecionado, o melhor local para instalar o Zeppelin é um **nó gateway**, quando o Cluster estiver com o firewall desligado e for protegido externamente.

- **Requisitos de Hardware**: **Mais memória e mais núcleos** sempre beneficiam o desempenho: recomenda-se um mínimo de 64 GB e 8 núcleos. Número de usuários: Um nó Zeppelin pode suportar de 8 a 10 usuários. Para mais usuários, várias instâncias podem ser configuradas.

- **Segurança**: Como qualquer software, **a segurança depende da matriz de ameaças e das opções de implantação**:

    - Autenticação
    - Kerberizar o Cluster usando Ambari

© Copyright Tecnisys 2026. All rights reserved.

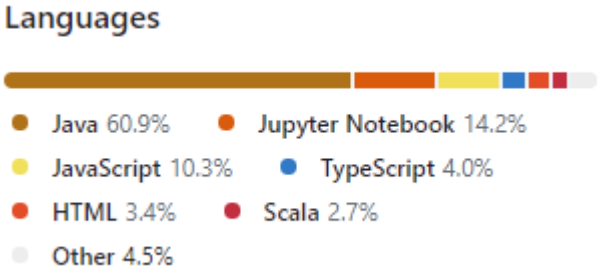


- Configure Zeppelin para alavancar o LDAP corporativo
- Não usar autenticação baseada em usuário local do Zeppelin, exceto para demonstrações.

- **Interpretadores**:
  - **Evite usar interpretador Shell**, pois o isolamento de segurança não é o ideal.
  - **Não use a IU do interpretador para representações**. Funciona apenas para interpretadores Livy e JDBC (Hive).
  - Os usuários devem **reiniciar suas próprias sessões** do interprete a partir do botão da **página do Notebook**, e **não na página do intérprete**, que reiniciaria as sessões para todos os usuários.
  - **Aproveite o interpretador Livy** para trabalhos do Spark no Cluster, pois ele fornece propagação de identidade ideal.
  - **Escolhendo os interpretadores**: Por default, Zeppelin registrará e mostrará todos os interpretadores abaixo da pasta $ZEPPELIN_HOME/interpreters. Mas existe a **possibilidade de especificar quais interpretadores devem ser incluídos ou excluídos** por meio das propriedades *zeppelin.interpreter.include* e *zeppelin_interpreter.exclude*. Apenas um deles pode ser especificado.
  - É possível criar um novo Interpreter, e a tarefa é simples. Basta estender a classe abstrata *org.apache.zeppelin.interpreter* e implementar alguns métodos.

Detalhes do Projeto Apache Zeppelin

O Apache Zeppelin é baseado em JVM, funcionando como um aplicativo web através do Jetty e permite que parágrafos dos notebooks sejam escritos em dúzias de linguagens diferentes, como Scala, Python, R, Markdown e SQL.



Linguagens do Zeppelin

© Copyright Tecnisys 2026. All rights reserved.

Fontes:

- Apache Zeppelin

© Copyright Tecnisys 2026. All rights reserved.

Apache ZooKeeper



Serviço de Coordenação Centralizada

Aplicações distribuídas consistem em múltiplos componentes de software que operam simultaneamente em diversos servidores físicos escaláveis, podendo abranger centenas ou milhares de máquinas.

Obviamente, este sistema está sujeito a falhas de hardware, travamentos, falhas de sistema, de comunicação e assim por diante, São falhas que não seguem um padrão e, portanto, tornam difícil aplicar um código de tolerância na lógica do aplicativo ou no projeto do Sistema.

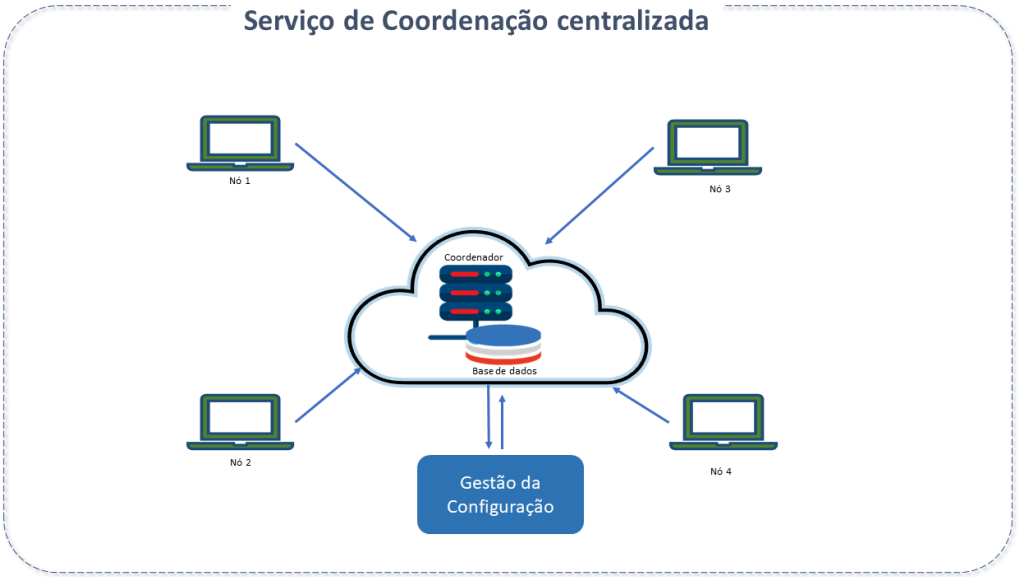
Além disso, fazer uma coordenação de *Cluster* correta, rápida e escalável é difícil e muitas vezes sujeita a erros, podendo gerar inconsistências no *Cluster*.

A coordenação centralizada garante a manutenção do "estado" consistente e da performance destes sistemas (como computadores e softwares), evitando alterações não documentadas no ambiente e auxiliando a evitar problemas de desempenho, inconsistências ou conformidade.

Executar essas tarefas manualmente é um trabalho complexo em sistemas grandes, como o Big Data. Pode envolver milhares de componentes para cada aplicativo.

Para isso, existem ferramentas que fornecem um plano e uma versão única e o estado desejado dos sistemas da organização, além de dar visibilidade de quaisquer modificações na configuração, por meio de trilhas de auditoria e rastreamento de alterações.

© Copyright Tecnisys 2026. All rights reserved.



Serviço de Coordenação Centralizada

Características do Apache Zookeeper

Apache Zookeeper é um projeto voluntário de software livre da Apache, originalmente desenvolvido pela YAHOO e, hoje, amplamente utilizado por grandes organizações como a própria Yahoo, Netflix e Facebook (uma lista completa de organizações e projetos que usam o Zookeeper pode ser vista aqui).

Trata-se de um serviço centralizado de código aberto, para coordenação de aplicações distribuídas.

Disponibiliza um conjunto de "primitivas" acessíveis por meio de APIs simples sobre as quais aplicações podem ser construídas a fim de implementar serviços de alto nível, tornando menos complicadas:

- Operações de associação de *Cluster* (detecção de saída ou junção de nós).
- Sincronização distribuída (bloqueios (*locks*) e barreiras).
- Nomeação (identificação de nós em um _Cluster pelo nome, semelhante ao DNS).
- Gerenciamento de configuração (informações de configuração mais recentes e atualizadas do sistema para um nó de junção).

A principal intenção do Apache Zookeeper é permitir que desenvolvedores de aplicativos se concentrem na lógica negocial e confiem inteiramente no Zookeeper para

© Copyright Tecnisys 2026. All rights reserved.



a correta coordenação:

- Ao contrário dos sistemas de arquivos convencionais, o Zookeeper fornece **alto rendimento e baixa latência**.
- Executa em uma coleção de máquinas e foi desenhado para **alta disponibilidade**, evitando a introdução de pontos de falha em sistemas.
- A ordem é muito importante. Sua **ordenação** permite que primitivas de sincronização sofisticadas sejam implementadas no cliente.

  Todas as atualizações são ordenadas. Cada atualização é marcada com um número que reflete esta ordem, chamado *zxid* (Zookeeper Transaction Id), que é exclusivo para cada atualização.

  As leituras (e relógios) são ordenadas em relação às atualizações. As respostas lidas são carimbadas com o último *zxid* processado pelo servidor que atende à leitura.
- Os aspectos de **desempenho** do Zookeeper permitem que seja usado em grandes sistemas distribuídos.

  O **desempenho** e a **escalabilidade** são alcançados por meio do mecanismo de watches, que permite que clientes se registrem para receber notificações sempre que um *znode* sofre alguma alteração (criação ou exclusão, mudanças nos dados mantidos pelo *znode*, criação ou exclusão de um dos filhos na sub-árvore de um _znode).
- Facilita interações com fraco acoplamento.

Arquitetura do Apache Zookeeper

Zookeeper é uma aplicação distribuída e altamente confiável de coordenação centralizada, seguindo o modelo cliente-servidor e operando em um conjunto de servidores replicados, conhecidos como ensemble, similarmente à gestão de serviços como o DNS.

O Zookeeper possui **arquitetura simples**, com namespace hierárquico padrão (que se assemelha ao filesystem Unix) de registradores de dados, denominados *znodes* (cada nó da árvore), o que o torna uma solução eficiente para gerenciamento de informações de configuração.

© Copyright Tecnisys 2026. All rights reserved.



Os **clientes** são nós que **consomem** os serviços e os **servidores** são nós que *fornecem* os serviços.

Os **caminhos** para os nós são expressos como caminhos **canônicos** (de acesso mais curto a um arquivo a partir do diretório raiz), absolutos e separados por barra.

Não há referência relativa.

A principal diferença entre Zookeeper e um sistema de arquivos padrão é que **todo *znode* pode ter dados associados como filhos** (todo arquivo pode ser também um diretório e vice-versa), e os *znodes* são limitados ao volume de dados que possuem.

Zookeeper foi desenhado para **armazenar dados de coordenação**: informações de *status*, configuração, informações de localização, etc.

Este tipo de meta-informação é normalmente medido em kilobytes, se não bytes.

Por isso, possui uma **checagem de "sanidade" integrada** de 1M, para evitar que seja usado como um grande "armazenador" de dados, mas, em geral, é usado para armazenar partes de dados bem menores.

Os principais componentes da arquitetura do Zookeeper são:

- **Clientes**: que se conectam ao serviço, a partir de APIs da biblioteca cliente Zookeeper (responsável pela interação de um aplicativo com o serviço Zookeeper), por meio de qualquer membro do *Ensemble*.

  Pode enviar e receber solicitações e respostas, bem como notificações e pulsações (*heartbeats*), por meio de uma conexão TCP.

  Se a conexão for interrompida, o cliente se conectará a um servidor diferente.

  Quando se conecta pela primeira vez, o primeiro servidor configurará uma sessão para o cliente.

  As solicitações de leitura são processadas localmente no servidor ao qual está conectado e atendidas a partir das réplicas locais de cada banco de dados do servidor.

© Copyright Tecnisys 2026. All rights reserved.

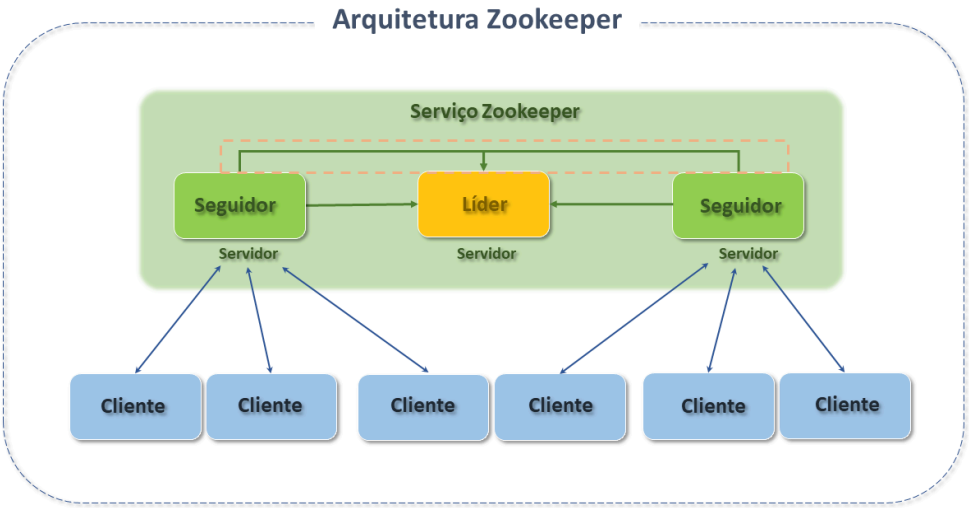
Se a solicitação registrar uma observação em um *znode*, esta também será rastreada localmente no servidor.

As solicitações de gravação, que alteram o estado do serviço, são processadas por um protocolo de "agreement" e encaminhadas para outros servidores ZooKeeper. Passam por consenso antes que uma resposta seja gerada.

As solicitações de sincronização também são encaminhadas para outro servidor, mas não passam por consenso.

- **Líder**: é o servidor que recupera automaticamente os nós com falhas. Todas as solicitações de gravação dos clientes são encaminhadas ao Líder.

- **Seguidores**: são os servidores não-líder, que recebem propostas de mensagens do líder e concordam com a entrega das mensagens.



Arquitetura Zookeeper

A camada de mensagens cuida da substituição de líderes em caso de falha e da sincronização entre seguidores e líderes.

O Zookeeper usa o **protocolo padrão de mensagens atômicas** para garantir a consistência dos dados entre os nós em todo o sistema. Isto permite garantir que as réplicas locais nunca divirjam.

Após receber a solicitação de alteração de dados, o líder grava o dado no disco e depois na memória.

© Copyright Tecnisys 2026. All rights reserved.



Zookeeper segue um **modelo de armazenamento compartilhado e hierárquico** similar a um sistema de arquivos tradicional. As abstrações providas pelo serviço permitem que os *znodes* sejam manipulados de forma simples e eficiente.

O serviço Zookeeper é **replicado no conjunto de máquinas que o compõem**.

Essas máquinas mantêm uma imagem *in memory* da árvore de dados junto com *logs* de transação e "snapshots" (instantâneos) em um armazenamento persistente. Uma vez que o dado é mantido *in memory*, alcança alto *throughput* (taxa de transferência) e baixa latência.

A desvantagem é que o tamanho do banco de dados que o Zookeeper pode gerenciar é limitado pela memória.

Essa limitação é mais um motivo para manter pequena a quantidade de dados armazenados em *znodes*.

Os servidores que compõem o serviço Zookeeper devem conhecer uns aos outros.

Enquanto a maioria estiver disponível, Zookeeper estará disponível. Os clientes também devem conhecer a lista de servidores, pois criam um identificador para o serviço usando esta lista.

Znodes

O namespace do Zookeeper é composto por registros de dados conhecidos como znodes, similares a arquivos e diretórios. O Znode mantém a estrutura de estatística que inclui número de versão para mudanças no dado, mudanças ACL, *timestamps*, para permitir validações de cache e atualizações coordenadas. Cada vez que os dados de um *znode* muda, o número da versão aumenta.

Existem dois tipos de *Znodes*:

- **Persistentes**: Os Znodes persistentes só deixam de compor o *namespace* quando são excluídos. Existem para armazenar dados que precisam estar altamente disponíveis e acessíveis por todos os componentes de um aplicativo distribuído.

- **Efêmeros**: Os *Znode* efêmeros são excluídos pelo Zookeeper quando a sessão do cliente termina, o que pode ocorrer por uma desconexão por falha ou um término de conexão. Embora estejam vinculados à sessão do cliente, são visíveis para todos

© Copyright Tecnisys 2026. All rights reserved.

os clientes, dependendo da política de **Lista de controle de acesso** (ACL) vinculada. Também podem ser excluídos pelo cliente criador ou qualquer outro autorizado.

Os *znodes* efêmeros podem ser usados para construir aplicativos distribuídos nos quais é necessário que os componentes conheçam o estado um dos outros ou de recursos constituintes.

- **Sequenciais**: Existe um terceiro nó, apontado por alguns como outro tipo de *znode* - o sequencial. Entretanto, este *znode* deve ser entendido como um qualificador dos dois principais.

> **NOTE**
>
> É atribuído um número de sequência pelo Zookeeper como parte de seu nome durante sua criação. O valor de um contador, mantido pelo *znode* pai é anexado ao nome.

Estes *znodes* são usados para implementação de uma fila global distribuída, pois os números de sequência podem impor uma ordenação. Também para projetar um serviço de bloqueio para um aplicativo distribuído.

Boas Práticas com Apache Zookeeper

Coisas a Evitar:

Existem alguns problemas que devem ser evitados por meio de uma correta configuração do Zookeeper:

- **Lista de servidores inconsistente**: A lista de servidores Zookeeper usada pelos clientes deve corresponder à lista de servidores Zookeeper que cada servidor Zookeeper possui. A lista cliente tem que ser um subconjunto da lista real. Além disso, as listas de servidores em cada arquivo de configuração dos servidores Zookeeper devem ser consistentes entre si.

- **Posicionamento incorreto do *log* de transação**: A parte mais crítica para o desempenho do Zookeeper é o *log* de transações. O Zookeeper sincroniza as transações com a mídia antes de retornar uma resposta. Um dispositivo de *log* de transações dedicado é a chave para um bom e consistente desempenho.

  Nunca devemos colocar o *log* em um dispositivo ocupado, pois isto prejudicará o desempenho. No caso de existir apenas um dispositivo de armazenamento,

© Copyright Tecnisys 2026. All rights reserved.



sugerimos colocar os arquivos de rastreamento no NFS e aumentar o *snapshotCount*. Isto não elimina o problema, mas pode mitigá-lo.

- **Swap para o disco**:
    - É necessário um cuidado especial na configuração do tamanho do heap java. Deve ser evitada a situação em que Zookeeper realize *swap* para disco. Tudo é ordenado e, portanto, se o processamento de uma solicitação "trocar" de disco, todas as outras solicitações na fila provavelmente farão o mesmo.
    - É recomendável ser conservador em estimativas: Com 4G de RAM, um tamanho máximo de heap do Java não deve ultrapassar 3G, por exemplo, pois o sistema operacional e o cache também precisam de memória.
    - A melhor prática é a execução de testes de carga, mantendo a garantia de estar bem abaixo do limite que poderia causar o *swap*.
    - O monitoramento de sistema como *vmstat* pode ser usado para monitorar estatísticas de memória virtual e decidir sobre o tamanho ideal de memória dependendo das necessidades do aplicativo. Em qualquer caso, sempre evitar o *swap*.

Coisas a fazer:

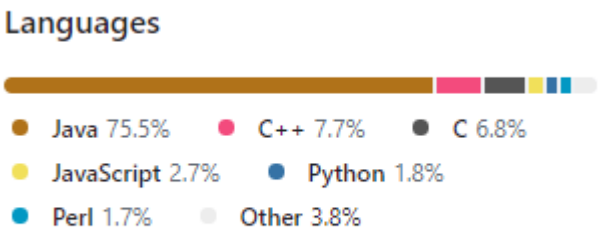
- É recomendável limpar o diretório de dados do Zookeeper periodicamente, caso a opção de autopurge não esteja habilitada. Este diretório contém o *snapshot* e os arquivos de *log* transacionais.
- Além disso, é importante avaliar a necessidade de backup, que, sendo o Zookeeper um serviço replicado, só é necessário em um dos servidores do conjunto.
- Zookeeper usa o Apache *log4j* como infraestrutura de registro. É recomendável definir o *rollover* automático usando o recurso *Log4j* embutido para *logs* do Zookeeper, à medida que os arquivos de *log* crescem.

Detalhes do Projeto Zookeeper

O Zookeeper é implementado em **Java**, e oferece **interfaces (bindings) para várias outras linguagens de programação** como C, Java, Perl e Python.

A lista completa de ligações-cliente disponíveis na comunidade pode ser acessada aqui.

© Copyright Tecnisys 2026. All rights reserved.



Linguagens do Zookeeper

Fontes:

- Wiki.Apache
- Zookeeper.apache.org
- Zookeeper GitHub

© Copyright Tecnisys 2026. All rights reserved.

PARTE II - TDP DATACENTER

© Copyright Tecnisys 2026. All rights reserved.

INSTALAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Requisitos Mínimos

Os requisitos mínimos para a implementação de um *Cluster de Big Data* podem variar de acordo com os serviços desejados ou as necessidades técnicas e organizacionais de cada ambiente.

Quantidade Mínima de Máquinas

Para atendermos minimamente os requisitos de replicação e alta disponibilidade dos principais componentes da Plataforma, sugerimos:

- 07 máquinas (*físicas ou virtuais*), sendo:
  - 03 máquinas dedicadas a componentes de controle, gestão e administração (*Masters*)
  - 03 máquinas dedicadas a componentes de armazenamento, operação e execução (*Workers*)
  - 01 máquina dedicada ao Apache Ambari e demais componentes utilitários (*Utility* ou *Edge*)

> **NOTE**
>
> *Clusters* com um número reduzido de máquinas podem ser implementados desde que sejam respeitados os requisitos técnicos, dimensionamento e capacidade de processamento de cada componente. Por exemplo, um *Cluster* exclusivo para o serviço do Apache Kafka pode ser implementado, inicialmente, com apenas 03 máquinas.

> **WARNING**
>
> É recomendado que as máquinas sejam de uso exclusivo do *Cluster de Big Data*.

A quantidade de recursos computacionais (CPU, RAM, disco, entre outros) indicados para cada máquina requer a avaliação de diversos fatores, tais como: serviços a serem instalados, cargas de trabalho desejadas (*workloads*), volume de dados esperados, entre outros.

© Copyright Tecnisys 2026. All rights reserved.



Caso necessite de apoio, entre em contato através da Área de Suporte.

> **NOTE**
>
> Para o download dos pacotes de instalação serão necessários 30GB de espaço livre em disco.

Virtualizadores Suportados

O TDP pode ser instalado em máquinas físicas ou virtuais. Os virtualizadores (*hypervisors*) suportados são:

- VMware vSphere Hypervisor (ESXi) 5.1 e versões superiores;
- Microsoft Hyper-V Server 2012 e versões superiores; e
- Oracle VirtualBox 5.0 e versões superiores.

Sistemas Operacionais Suportados

Atualmente, o TDP está disponível para os seguintes sistemas operacionais:

- CentOS 8.x e 9.x;
- Red Hat Enterprise Linux 8.x e 9.x;
- Rocky Linux 8.x e 9.x; e
- AlmaLinux 8.x e 9.x.

Em breve, novas distribuições Linux estarão suportadas.

Navegadores Suportados

Para acessar as interfaces gráficas dos componentes de um *Cluster TDP* recomendamos o uso dos seguintes navegadores:

- Google Chrome 96 ou versões superiores;
- Mozilla Firefox 94 ou versões superiores; e
- Microsoft Edge 96 ou versões superiores.

Lista de Verificação

Antes de prosseguir para a instalação, confirme todos os itens abaixo:

© Copyright Tecnisys 2026. All rights reserved.

- [ ] Mínimo de 7 máquinas disponíveis (3 Masters, 3 Workers, 1 Utility/Edge)
- [ ] Sistema operacional suportado instalado (CentOS, RHEL, Rocky Linux ou AlmaLinux 8.x/9.x)
- [ ] 30GB de espaço livre em disco para download dos pacotes
- [ ] Firewall desativado ou regras configuradas
- [ ] SELinux desativado permanentemente
- [ ] Parâmetros do kernel configurados (swappiness, overcommit_memory, THP)
- [ ] Serviço de sincronização de tempo (NTP/Chrony) operacional
- [ ] Conectividade de rede e resolução DNS entre todas as máquinas
- [ ] Acesso SSH configurado entre as máquinas
- [ ] Credenciais de acesso aos repositórios TDP obtidas

© Copyright Tecnisys 2026. All rights reserved.

Ferramentas de Apoio

As ferramentas indicadas abaixo podem facilitar as atividades de implantação e administração do *Cluster de Big Data*:

- **epel release** - Conjunto de pacotes de código aberto que fornece diversos software complementares para distribuições Enterprise Linux, como ferramentas de administração, desenvolvimento, monitoramento, entre outras atividades.
- **telnet** - Ferramenta de rede que possibilita a conexão remota entre duas máquinas, auxiliando no teste e correção de problemas de comunicação entre servidores.
- **net tools** - Pacote com diversas ferramentas de rede úteis (e.g. *netstat*) para escanear faixas de IPs, análise de rotas, "ping", identificação de conexões, entre outras atividades.
- **wget** - Ferramenta utilizada para a requisição, ou *download*, de conteúdo e arquivos de servidores na internet. Suporta *downloads* via FTP, SFTP, HTML e HTTPS.
- **vim** - Editor de texto versátil. Uma versão aprimorada do editor de texto "vi".

Instruções

Para instalar as ferramentas de apoio, execute o comando abaixo:

Terminal input

```
yum install epel-release telnet net-tools wget vim
```

© Copyright Tecnisys 2026. All rights reserved.

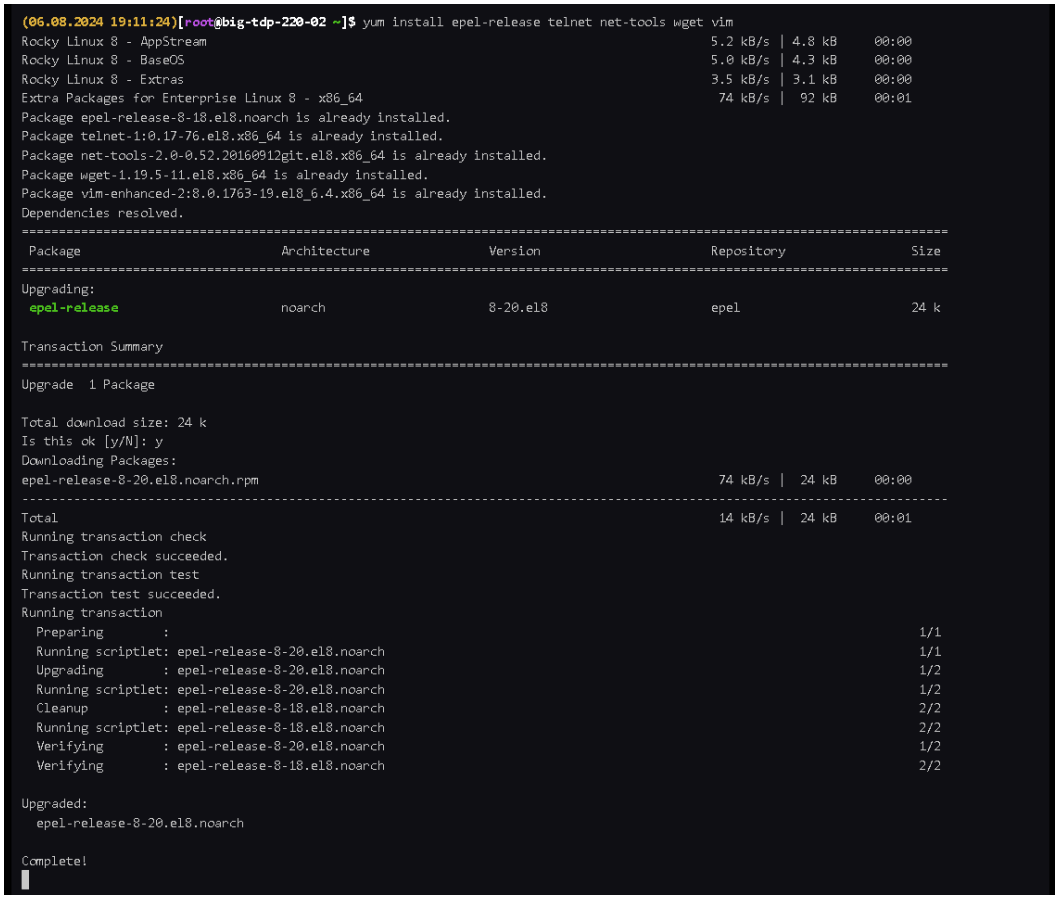


Figure 2 - Instalação das ferramentas de apoio

© Copyright Tecnisys 2026. All rights reserved.

Preparação do Ambiente

Os procedimentos a seguir deverão ser realizados em todas as máquinas do *Cluster de Big Data*.

Desativação do Firewall

Instruções

1. Desabilite o serviço do Firewall:

Terminal input

```
systemctl stop firewalld;
systemctl disable firewalld;
```

> **WARNING**
>
> Não sendo possível a desativação do Firewall, crie regras (iptables) para liberar a comunicação entre as máquinas nas portas utilizadas pelos serviços da Plataforma TDP.

Desativação do SELinux

Instruções

1. Desative temporariamente o SELinux:

Terminal input

```
setenforce 0;
```

2. Desative permanentemente o SELinux:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
sed -i --follow-symlinks "s+SELINUX=enforcing+SELINUX=disabled+g"
/etc/selinux/config;
```

3. Reinicie a máquina para efetivar a desativação permanente do SELinux.

Configuração de parâmetros do Kernel

Instruções

1. Evite ao máximo o uso da área de swap:

Terminal input

```
echo 'vm.swappiness=1' >> /etc/sysctl.conf;
sysctl -p /etc/sysctl.conf;
```

> **NOTE**
>
> O kernel do Linux fornece uma configuração ajustável que controla a frequência com que a área de swap em disco é usada, chamada *swappiness*. Uma configuração de *swappiness* igual a zero significa que o disco será evitado a menos que seja absolutamente necessário (quando o servidor ficar sem memória), enquanto uma configuração de *swappiness* igual a 100 significa que os programas utilizarão a área de swap quase instantaneamente. Reduzir o valor de *swappiness* reduz a probabilidade de o kernel do Linux enviar a memória de uma aplicação para a área de swap. A área de swap é extremamente mais lenta que a memória, pois utiliza o disco em vez da memória RAM. Quando a área de memória dos processos é transferida para o disco eles podem sofrer pausas, o que pode causar problemas em serviços, como por exemplo o Apache Zookeeper.

© Copyright Tecnisys 2026. All rights reserved.

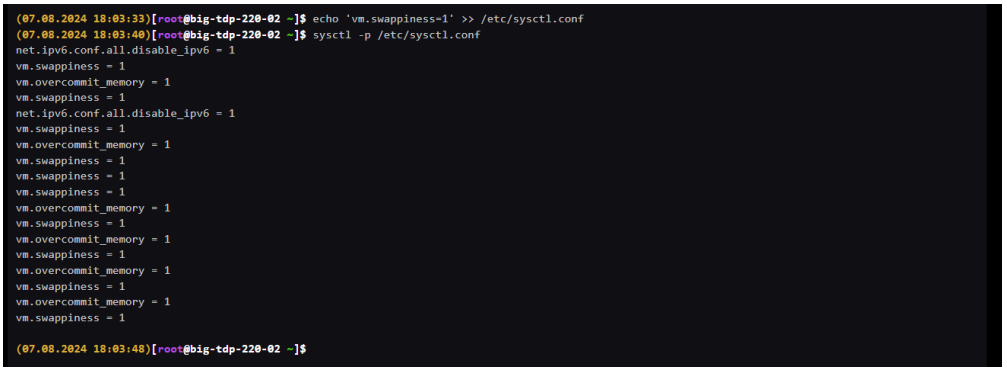


Figure 1 - Minimiza swap

2. Altere o comportamento padrão de alocação de memória RAM:

Terminal input

```
echo 'vm.overcommit_memory=1' >> /etc/sysctl.conf;
sysctl -p /etc/sysctl.conf;
```

> **NOTE**
>
> Em ambientes de *Big Data*, nos quais é comum termos máquinas com grandes quantidades de memória RAM, recomendamos o valor 1 para a configuração *overcommit_memory*. Diferentemente do valor 0, que utiliza uma abordagem heurística para atender requisições de memória (*malloc*), o valor 1 assume que sempre há memória física suficiente, aumentando consideravelmente o desempenho de tarefas de uso intenso de memória.



Figure 2 - Muda default RAM

3. Desabilite o Transparent Huge Pages (THP):

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled;
echo never > /sys/kernel/mm/transparent_hugepage/defrag;
echo "echo never > /sys/kernel/mm/transparent_hugepage/enabled" >>
/etc/rc.local;
echo "echo never > /sys/kernel/mm/transparent_hugepage/defrag" >>
/etc/rc.local;
```

NOTE

Muitas distribuições Linux disponibilizam o THP como uma opção de baixa complexidade para aumentar o tamanho de blocos/páginas de memória (de 4KB para 2MB ou 1GB) e possibilitar o gerenciamento de muitos gigabytes, e até terabytes, de memória RAM. No entanto, as cargas de trabalho em ambientes de *Big Data* costumam ter um desempenho ruim com o THP ativado, porque tendem a ter padrões de acesso à memória esparsos, em vez de contínuos, sobrecarregando assim o CPU.

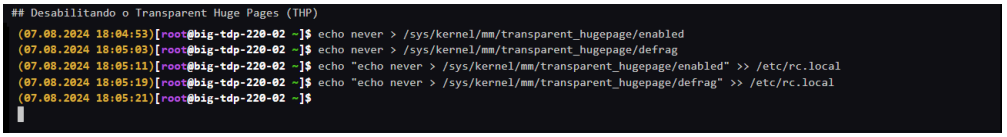


Figure 3 - Desabilita THP

Sincronização de Tempo

Instruções

1. Instale um serviço de sincronização de tempo (NTP) para assegurar que a data e a hora das máquinas estejam sempre sincronizadas, prevenindo inconsistências em dados e serviços. Neste exemplo, utilizaremos o Chrony:

Terminal input

```
yum install chrony -y
systemctl enable chronyd
systemctl start chronyd
```

© Copyright Tecnisys 2026. All rights reserved.

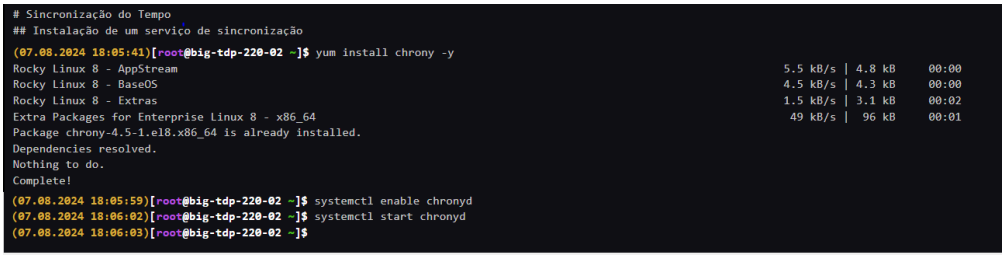


Figure 4 - Instala a sincronização do tempo

2. Configure o Chrony para sincronizar com servidores NTP apropriados:

Terminal input

```
vi /etc/chrony.conf

# Adicione ou ajuste os servidores NTP conforme necessário

server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
```

> **NOTE**
>
> Para salvar e sair do editor vi, pressione ESC, digite :wq e pressione Enter.

3. Reinicie o serviço para aplicar as novas configurações:

Terminal input

```
systemctl restart chronyd
```

4. Verifique o status da sincronização:

Terminal input

```
chronyc tracking
chronyc sources
```

© Copyright Tecnisys 2026. All rights reserved.

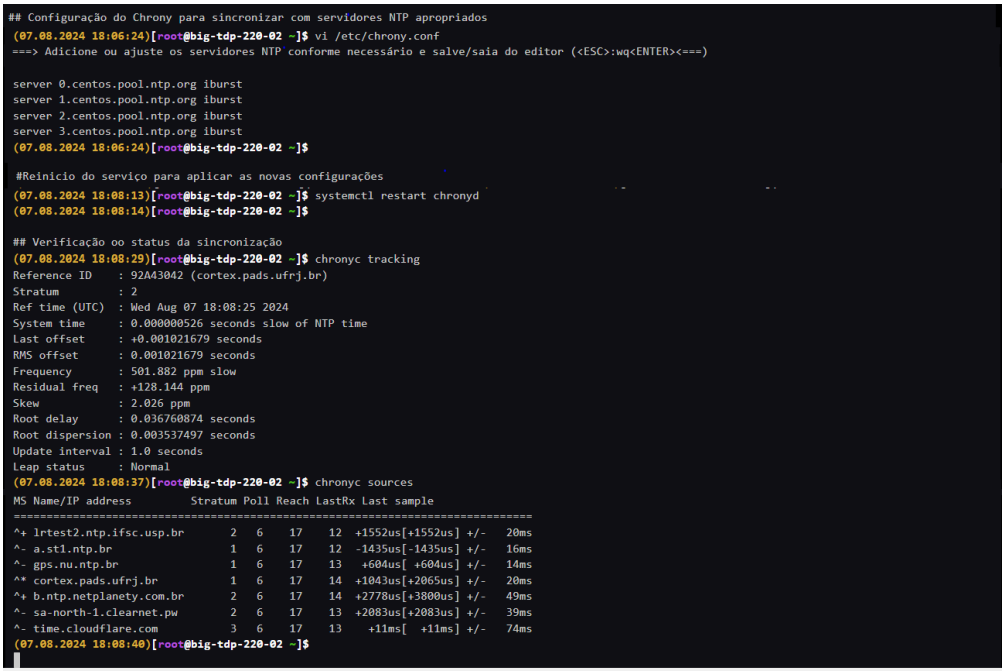


Figure 5 - Configura Chrony

© Copyright Tecnisys 2026. All rights reserved.

Pacotes de Instalação

Todos os pacotes de instalação do TDP, incluindo dependências diretas, estão disponíveis no Repositório Público de Pacotes da Tecnisys. Para acessá-los, utilize as credenciais de acesso definidas no momento do seu cadastro *gratuito* no site da Tecnisys.

> **NOTE**
>
> Em uma implantação com acesso limitado ou restrito à Internet, é possível realizar o download dos arquivos necessários e criar um repositório de pacotes local. Para isso, siga os passos a seguir. Caso contrário, siga para Download do Script de Instalação.

Download dos Pacotes

Instruções

Para baixar os pacotes de instalação, siga os passos abaixo:

1. Faça o seu cadastro gratuito no site da Tecnisys;
2. Acesse o Repositório Público de Pacotes da Tecnisys;
3. Clique no botão *Sign out*, localizado no canto superior direito da página;
4. Informe as suas credenciais de acesso (as mesmas do site da Tecnisys);
5. Clique na opção *Browse*, localizada no menu lateral esquerdo; e
6. Na área de navegação central, acesse o diretório desejado para download dos arquivos.

© Copyright Tecnisys 2026. All rights reserved.

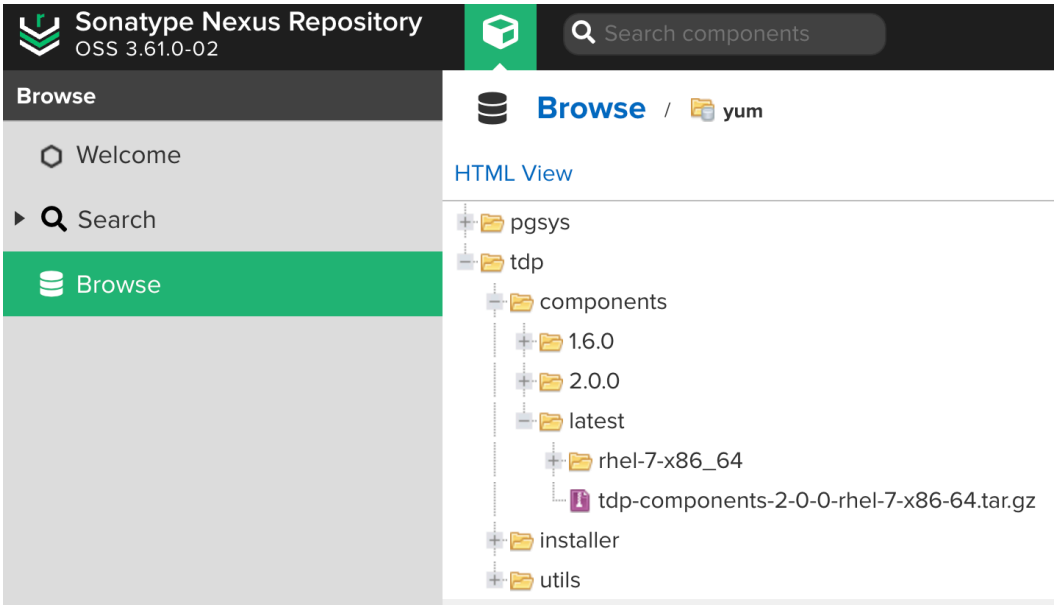


Figura 1 - Repositório Público de Pacotes da Tecnisys

Também é possível baixar os pacotes diretamente do terminal, desde que sejam informadas as credenciais de acesso na requisição:

- Exemplo via curl

Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/tdp-ambari-3-0-el-9-x86-
64.tar.gz -o tdp-ambari-3-0-el-9-x86-64.tar.gz
```

Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/components/3.0/tdp-components-3-0-el-9-
x86-64.tar.gz -o tdp-components-3-0-el-9-x86-64.tar.gz
```

Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/utils/3.0/tdp-utils-3-0-el-9-x86-
64.tar.gz -o tdp-utils-3-0-el-9-x86-64.tar.gz
```

© Copyright Tecnisys 2026. All rights reserved.

- Exemplo via wget

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/tdp-ambari-3-0-el-9-x86-
64.tar.gz
```

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/components/3.0/tdp-components-3-0-el-9-
x86-64.tar.gz
```

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/utils/3.0/tdp-utils-3-0-el-9-x86-
64.tar.gz
```

Lembre-se de substituir *USUARIO* e *SENHA* por, respectivamente, o seu usuário e a sua senha cadastrados no site da Tecnisys.

Todos os pacotes de instalação de componentes e utilitários podem ser baixados de uma só vez através de arquivos compactados (tar.gz). Por exemplo:

Terminal input

```
/yum/tdp/ambari/3.0/tdp-ambari-3-0-el-9-x86-64.tar.gz
```

Terminal input

```
/yum/tdp/components/3.0/tdp-components-3-0-el-9-x86-64.tar.gz
```

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
/yum/tdp/utils/3.0/tdp-utils-3-0-el-9-x86-64.tar.gz
```

TIP

É possível verificar a integridade do arquivo conferindo os atributos de Checksum (sha512, sha256, sha1 e md5), exibidos ao clicá-lo.

INFO

Para uma instalação local, não se esqueça de baixar e importar a chave pública GPG dos repositórios (RPM-GPG-KEY-TDP) disponível em:

Terminal input

```
/yum/tdp/ambari/3.0/el-9-x86_64/RPM-GPG-KEY
```

Terminal input

```
/yum/tdp/components/3.0/el-9-x86_64/RPM-GPG-KEY
```

Terminal input

```
/yum/tdp/utils/3.0/el-9-x86_64/RPM-GPG-KEY
```

Os arquivos compactados (tar.gz) dos pacotes de instalação de componentes e utilitários já contemplam a chave pública GPG.

Criação de um Repositório de Pacotes Local

Instruções

© Copyright Tecnisys 2026. All rights reserved.

Em uma implantação sem acesso, ou com acesso restrito, à Internet, faça, previamente, o download dos pacotes de instalação através de uma máquina com acesso à Internet, e transfira tais arquivos para o seu ambiente de *Big Data*.

Caso ainda não exista um repositório de pacotes local, é possível criá-lo da seguinte forma:

1. Instale as ferramentas HTTPD e createrepo:

Terminal input

```
yum install httpd createrepo
```

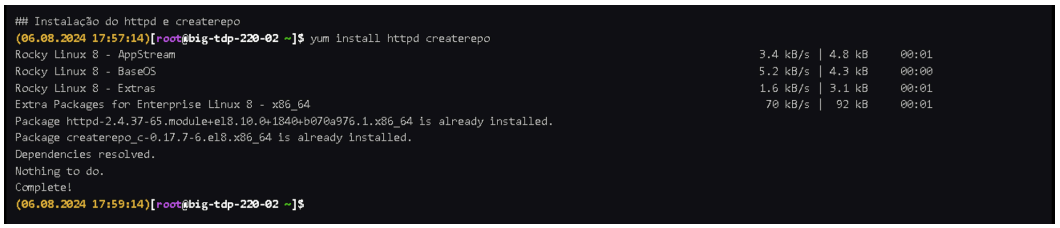


Figura 2 - Instalação createrepo e httpd

2. Crie o diretório do repositório de pacotes (ambari, components e utils). Por exemplo:

Terminal input

```
mv /tmp/tdp-ambari-3-0-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-ambari-3-0-el-9-x86-64.tar.gz -C /var/www/html
```

Terminal input

```
mv /tmp/tdp-components-3-0-el-9-x86-64.tar.gz /var/www/html
```

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
tar -xvzf /var/www/html/tdp-components-3-0-el-9-x86-64.tar.gz -C /var/www/html
```

Terminal input

```
mv /tmp/tdp-utils-undefined-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-utils-3-0-el-9-x86-64.tar.gz -C /var/www/html
```

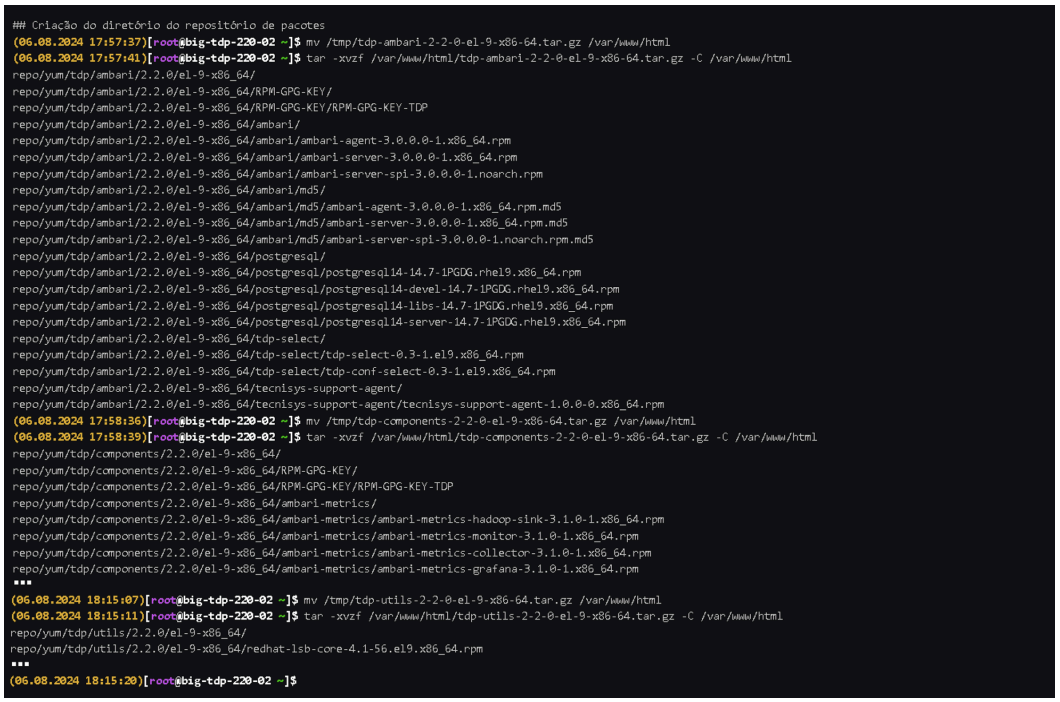


Figura 3 - Criação diretorio do repositorio

3. Crie o índice de pacotes do repositório:

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/ambari/3.0/el-9-x86_64/
```

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/components/3.0/el-9-x86_64/
```

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/utils/3.0/el-9-x86_64/
```

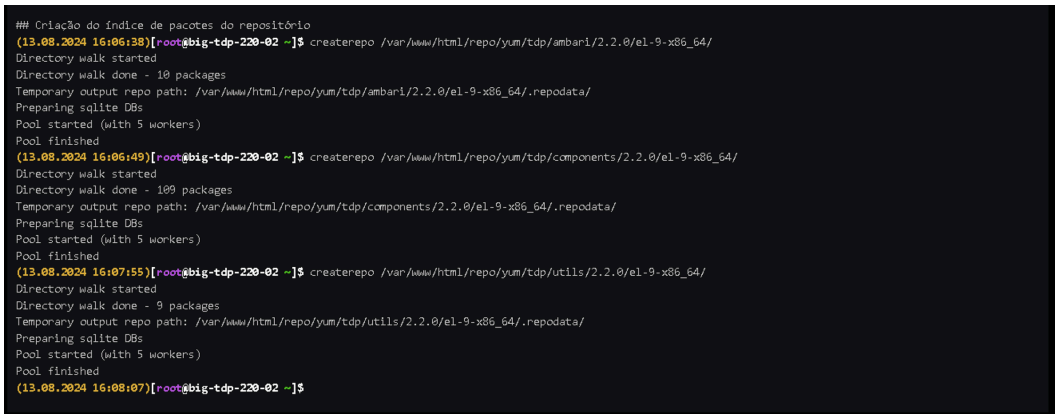


Figura 4 - Criação do índice de pacotes

4. Adicione o alias do repositório no arquivo de configuração do serviço HTTPD:

Terminal input

```
cat << EOF   /etc/httpd/conf/httpd.conf
    Alias /repo /var/www/html/repo
    <Directory /var/www/html/repo>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    EOF
```

© Copyright Tecnisys 2026. All rights reserved.



Figura 5 - Adiciona alias do repositorio no arquivo de configuração do HTTPD

5. Reinicie o serviço HTTPD:

Terminal input

```
systemctl restart httpd
```



Figura 6 - Reinicia Serviço

Confirme a disponibilidade e a organização dos pacotes de instalação acessando o endereço `http://localhost/repo`. Substitua ´localhost´ pelo IP ou *hostname* da máquina do repositório de pacotes caso esteja realizando o acesso de outra máquina.

Download do Script de Instalação

Instruções

Desenvolvemos um *shell script* parametrizável para auxiliá-lo na primeira etapa do processo de implantação do *Cluster de Big Data*, a instalação do componente Apache Ambari.

> **NOTE**

© Copyright Tecnisys 2026. All rights reserved.



> Recomendamos que a instalação do Apache Ambari (Ambari Server e Ambari Web) seja realizada em uma máquina do tipo *Utility* ou *Edge*.

Tal script está disponível no diretório `/yum/tdp/installer` do Repositório Público de Pacotes da Tecnisys.

Inicialmente, o script deve ser baixado na máquina onde será instalado o Ambari Server. Porém, caso o registro das máquinas que irão compor o *Cluster de Big Data* venha a ser manual (sem a troca de chaves SSH), conforme descrito em Registro de Hosts, é recomendável que o script em questão seja baixado em todas as máquinas do *Cluster* para instalação do Ambari Agent.

É possível baixar o script de instalação diretamente do terminal, desde que sejam informadas as credenciais de acesso na requisição:

- **Exemplo via curl**

  Terminal input

  ```
  curl -S --user USUARIO:SENHA
  https://repo.tecnisys.com.br/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-
  installer-el-9.sh -o ambari-tdp-installer-el-9.sh
  ```

- **Exemplo via wget**

  Terminal input

  ```
  wget --user USUARIO --password SENHA
  https://repo.tecnisys.com.br/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-
  installer-el-9.sh
  ```

Lembre-se de substituir *USUARIO* e *SENHA* por, respectivamente, o seu usuário e a sua senha cadastrados no site da Tecnisys.

© Copyright Tecnisys 2026. All rights reserved.

Instalação do Apache Ambari

Estando os requisitos mínimos atendidos, todas as máquinas do ambiente preparadas e os pacotes de instalação disponíveis, seguimos para a instalação do serviço do Apache Ambari.

Após instalado, o Apache Ambari será responsável pela criação do *Cluster de Big Data*, seja através de sua página web ou via REST API.

Instalação via script

Execute o comando abaixo para instalar, via script de instalação, o Ambari Server em uma máquina *utility* ou *edge* [1]:

Terminal input

```
    sh ambari-tdp-installer-el-9.sh -b URL_BASE_PARA_OS_PACOTES_DOS_COMPONENTES
-u USUARIO -p SENHA;
```

Os parâmetros USUÁRIO (-u) e SENHA (-p) são opcionais e só devem ser informados quando o repositório de pacotes requisitar credenciais de acesso, como é o caso do Repositório Público de Pacotes da Tecnisys.

> **NOTE**
>
> Seguem as opções disponíveis do script de instalação ambari-tdp-installer-el-9.sh:
>
> - **-b, --baseurl** URL base para os pacotes dos componentes. Default:
>
>   Terminal input
>
>   ```
>   https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/el-9-x86_64
>   ```
>
> - **-u, --username** Usuário para acesso ao repositório de pacotes. Deve ser informado ao se utilizar a URL do Repositório de Pacotes da Tecnisys.

© Copyright Tecnisys 2026. All rights reserved.



- **-p, --password** Senha para acesso ao repositório de pacotes. Deve ser informada ao se utilizar a URL do Repositório de Pacotes da Tecnisys.
- **-c, --component (agent | server | all)** [default: `server`] Componente do Apache Ambari a ser instalado.

Cenários de Instalação do Ambari Server

Instruções

A seguir, apresentamos exemplos para diferentes cenários:

- **Instalação usando o Repositório de Pacotes Públicos da Tecnisys**:

  Terminal input

  ```
  sh /repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --username "user" --password "pass"
  ```

  As credenciais de acesso *(username e password)* são definidas ao se cadastrar gratuitamente no Site da Tecnisys

- **Instalação usando um repositório de pacotes local:**

  Terminal input

  ```
  sh /repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --baseurl "file:///opt/repo/tdp/ambari/3.0/el-9-x86_64"
  ```

- **Instalação usando um repositório de pacotes local na máquina de IP 192.168.32.100 e acessível via HTTP:**

  Terminal input

  ```
  /repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --baseurl "https://192.168.32.100/tdp/ambari/3.0/el-9-x86_64"
  ```

© Copyright Tecnisys 2026. All rights reserved.

Instalação via tdpctl

A instalação do Ambari Server também pode ser realizada através do tdpctl, a Interface de Linha de Comando do TDP:

Terminal input

```
tdpctl install --help
```

As opções disponíveis, assim como as informações necessárias para a instalação do Ambari Server, são as mesmas solicitadas pelo script de instalação.

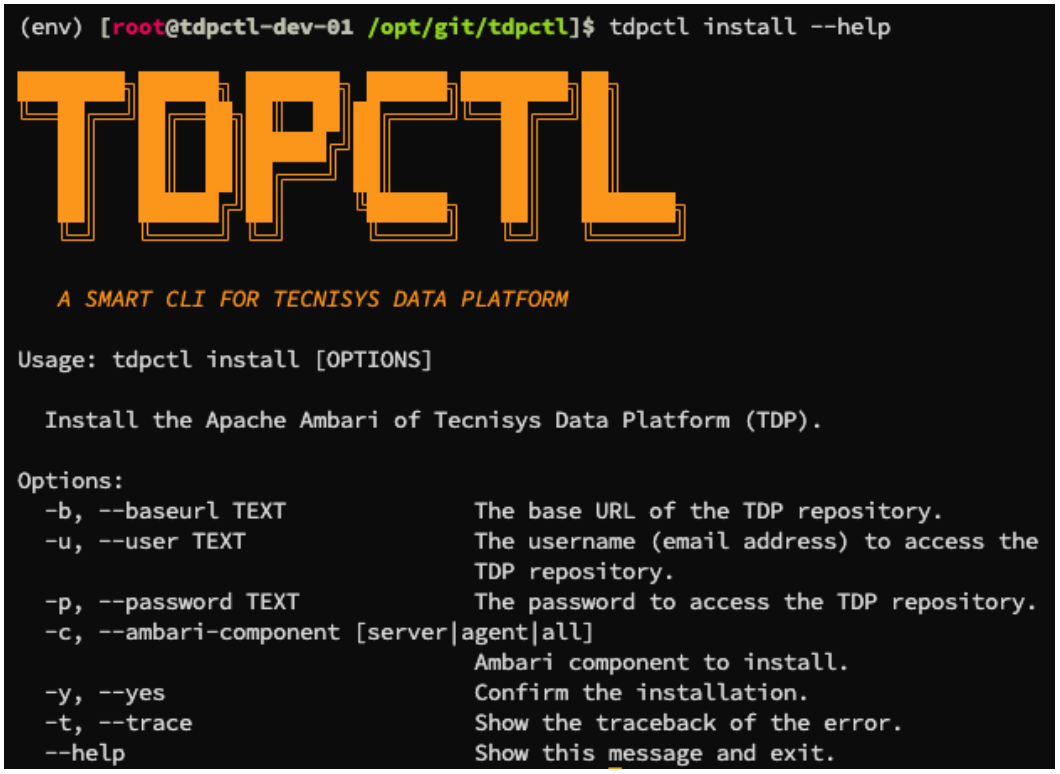


Figura 1 - Opções para o comando tdpctl install

Note que é possível informar o componente desejado através da opção -c ou --component (agent | server | all).

© Copyright Tecnisys 2026. All rights reserved.



Figura 2 - Instalação do Ambari Server via tdpctl

Para instalar o tdpctl, siga as orientações disponíveis aqui.

Configurações

Instruções

Configuração do driver JDBC

Configure o driver JDBC a ser utilizado pelo Ambari Server para se conectar no banco de metadados:

Terminal input

```
ambari-server setup --jdbc-db=postgres --jdbc-driver=postgresql-42.2.16.jar
```

© Copyright Tecnisys 2026. All rights reserved.

> **TIP**
>
> O driver JDBC do PostgreSQL também está disponível no diretório public/ambari/3.0.0.4 do Repositório de Pacotes Tecnisys

```
# Configuração do JDBC
(14.08.2024 19:38:16)[root@big-tdp-220-02 ~]$ ambari-server setup --jdbc-db=postgres --jdbc-driver=/tmp/postgresql-42.2.16.jar
Using python
Setup ambari-server
Copying /tmp/postgresql-42.2.16.jar to /var/lib/ambari-server/resources/postgresql-42.2.16.jar
Creating symlink /var/lib/ambari-server/resources/postgresql-42.2.16.jar to /var/lib/ambari-server/resources/postgresql-jdbc.jar
If you are updating existing jdbc driver jar for postgres with postgresql-42.2.16.jar. Please remove the old driver jar, from all hosts. Restarting service
s that need the driver, will automatically copy the new jar to the hosts.
JDBC driver was successfully initialized.
Ambari Server 'setup' completed successfully.
```

Figura 3 - Configuração do JDBC Server

```
# Ambari Server Configuration
(14.08.2024 19:39:00)[root@big-tdp-220-02 ~]$ ambari-server setup
Using python
Setup ambari-server
Checking SELinux...
SELinux status is 'disabled'
SELinux status is 'disabled'
Customize user account for ambari-server daemon [y/n] (n)? n
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Checking JDK...
[1] OpenJDK 64-Bit Server VM (Temurin)(build 25.402-b06, mixed mode)
[2] Custom JDK
================================================================================
Enter choice (1):
To download the Open JDK Policy Files you must accept the license terms found at https://github.com/openjdk/jdk/blob/master/LICENSE and not accepting will
cancel the Ambari Server setup and you must install the JDK and JCE files manually.
Do you accept the Code License Agreement [y/n] (y)? y
Downloading JDK from https://repo.tecnisys.com.br/repository/public/ambari/3.0.0.0/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz to /var/lib/ambari-serve
r/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz... 100% (98.2 MB of 98.2 MB)
Successfully downloaded JDK distribution to /var/lib/ambari-server/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
Installing JDK to /usr/lib/jvm/
Successfully installed JDK to /usr/lib/jvm/
Downloading JCE Policy archive from https://repo.tecnisys.com.br/public/ambari/3.0.0.0/jce_policy-8.zip to /var/lib/ambari-server/resources/jce_policy-8.zi
p

Successfully downloaded JCE Policy archive to /var/lib/ambari-server/resources/jce_policy-8.zip
Installing JCE policy...
Check JDK version for Ambari Server...
JDK version found: 8
Minimum JDK version is 8 for Ambari. Skipping to setup different JDK for Ambari Server.
Checking GPL software agreement...
GPL License for LZO: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
Enable Ambari Server to download and install GPL Licensed LZO packages [y/n] (n)? y
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Configuring PostgreSQL...
Restarting PostgreSQL
Creating schema and user...
done.
Creating tables...
done.
Extracting system views...
ambari-admin-3.0.0.0-1.jar
ambari-views-package-3.0.0.0-1.jar
capacity-scheduler-3.0.0.0-1.jar
files-3.0.0.0-1.jar
pig-3.0.0.0-1.jar
wfmanager-3.0.0.0-1.jar

Ambari repo file doesn't contain latest json url, skipping repoinfos modification
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
```

Figura 4 - Configuração do Ambari Server

Configuração do Ambari Server

Configure o Ambari Server:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
ambari-server setup
```

Na sequência, forneça as informações solicitadas no prompt apresentado:

1- Se SELinux estiver ativo, será apresentado um alerta. Confirme se o SELinux pode ser alterado para o modo permissivo e temporariamente desativado. Default (y)

2- Confirme se deseja utilizar um usuário de SO customizado para o daemon do Ambari Server. Default (n)

> (a) - Se sim (y), informe o usuário desejado.

3- Se o Firewall (iptables) estiver ativo, será apresentado um alerta. Confirme se as portas utilizadas pelo Ambari estão acessíveis. Default (y)

> (a) - Caso haja conflito de porta, será apresentado um erro e a configuração abortada.

4- Escolha a JDK a ser utilizada. Default (1) *Oracle JDK 1.8 + Java Cryptography Extension (JCE) Policy Files 8* (baixados automaticamente do diretório *public/tdp/2.0.0/rhel-7-x86_64* do Repositório Público de Pacotes da Tecnisys.

> (a) - Se escolhida a opção (1), confirme se você aceita a licença de uso. Default (y)

> (b) - Escolha a opção (2) para informar o caminho do JAVA_HOME de uma JDK diferente.

5- Confirme se deseja que o Ambari Server baixe e instale pacotes adicionais de compressão LZO. Default (n)

6- Confirme se deseja prosseguir com as Configurações Avançadas de Banco de Dados. Default (n)

> (a) - Se não (n), automaticamente o Ambari inicializa uma instância do PostgreSQL (dependência de instalação do Ambari Server), cria o seu próprio banco de metadados, esquema, usuário, tabelas de metadados, etc.

© Copyright Tecnisys 2026. All rights reserved.

(b) - Se sim (y), serão solicitadas informações de conexão ao banco de dados, tais como, o Sistema Gerenciador de Banco de Dados (SGBD), hostname, porta e nome do banco de dados do Ambari, para que o Ambari Server se conecte e crie automaticamente as suas tabelas de metadados entre outras estruturas.

```
# Ambari Server Configuration
(14.08.2024 19:39:00)[root@big-tdp-220-02 ~]$ ambari-server setup
Using python
Setup ambari-server
Checking SELinux...
SELinux status is 'disabled'
SELinux status is 'disabled'
Customize user account for ambari-server daemon [y/n] (n)? n
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Checking JDK...
[1] OpenJDK 64-Bit Server VM (Temurin)(build 25.402-b06, mixed mode)
[2] Custom JDK
==============================================================================
Enter choice (1):
To download the Open JDK Policy Files you must accept the license terms found at https://github.com/openjdk/jdk/blob/master/LICENSE and not accepting will
cancel the Ambari Server setup and you must install the JDK and JCE files manually.
Do you accept the Code License Agreement [y/n] (y)? y
Downloading JDK from https://repo.tecnisys.com.br/repository/public/ambari/3.0.0.0/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz to /var/lib/ambari-serve
r/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz... 100% (98.2 MB of 98.2 MB)
Successfully downloaded JDK distribution to /var/lib/ambari-server/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
Installing JDK to /usr/lib/jvm/
Successfully installed JDK to /usr/lib/jvm/
Downloading JCE Policy archive from https://repo.tecnisys.com.br/public/ambari/3.0.0.0/jce_policy-8.zip to /var/lib/ambari-server/resources/jce_policy-8.zi
p

Successfully downloaded JCE Policy archive to /var/lib/ambari-server/resources/jce_policy-8.zip
Installing JCE policy...
Check JDK version for Ambari Server...
JDK version found: 8
Minimum JDK version is 8 for Ambari. Skipping to setup different JDK for Ambari Server.
Checking GPL software agreement...
GPL License for LZO: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
Enable Ambari Server to download and install GPL Licensed LZO packages [y/n] (n)? y
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Configuring PostgreSQL...
Restarting PostgreSQL
Creating schema and user...
done.
Creating tables...
done.
Extracting system views...
ambari-admin-3.0.0.0-1.jar
ambari-views-package-3.0.0.0-1.jar
capacity-scheduler-3.0.0.0-1.jar
files-3.0.0.0-1.jar
pig-3.0.0.0-1.jar
wfmanager-3.0.0.0-1.jar

Ambari repo file doesn't contain latest json url, skipping repoinfos modification
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
```

Figure 4 - Configuração do Ambari Server

Inicialização do Ambari Server

Inicie o daemon do Ambari Server:

Terminal input

```
ambari-server start
```

© Copyright Tecnisys 2026. All rights reserved.

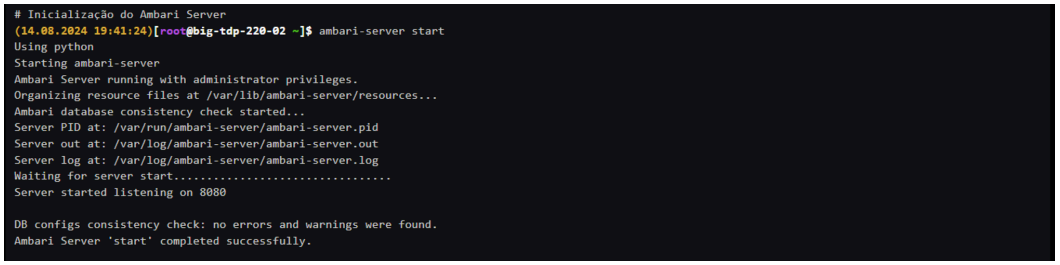


Figura 5 - Inicialização do Ambari Server

Após a inicialização do Ambari Server, a página web do Ambari estará, por padrão, acessível na porta 8080.

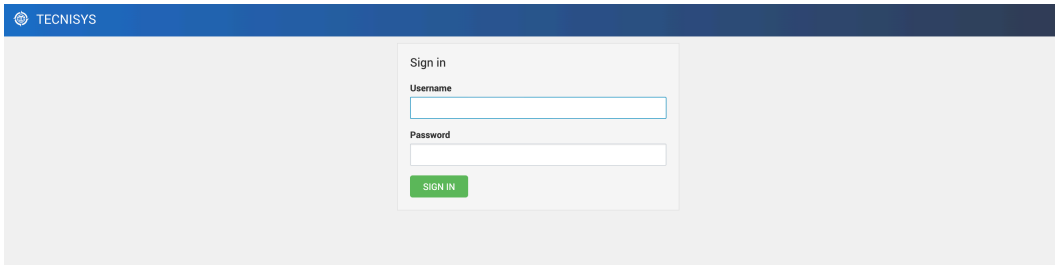


Figura 6 - Página de login do Ambari

> **NOTE**
>
> Se necessário, a porta da interface web do Ambari pode ser alterada no arquivo `/etc/ambari-server/conf/ambari.properties`.

Criação dos Bancos de Metadados

Muitos dos serviços/componentes da Plataforma TDP necessitam de bancos de dados para a persistência de metadados.

A seguir, fornecemos orientações para a criação prévia desses bancos de dados no PostgreSQL (SGBD padrão do Ambari Server). Caso tenha escolhido um SGBD diferente para o seu ambiente, consulte a documentação oficial do mesmo.

> **WARNING**
>
> As senhas utilizadas nos comandos a seguir visam apenas exemplificar o procedimento apresentado. Por gentileza, utilize senhas fortes em seu

© Copyright Tecnisys 2026. All rights reserved.

ambiente. Além disso, os nomes dos usuários e bancos de dados são apenas sugestões, fique à vontade para alterá-los.

> **CAUTION**
>
> Atenção para a collation (configurações de locale e encoding, por exemplo: en_US.UTF-8 e pt_BR.UTF-8) dos bancos de metadados. Alguns componentes podem não funcionar corretamente ou apresentar problemas na exibição, comparação e ordenção de dados e informações se a collation não estiver configurada corretamente.

Criação do Banco de Metadados do Apache Airflow

1. Crie o usuário airflow:

Terminal input

```
su -c 'createuser -s -d -l airflow' - postgres;
```

2. Defina a senha do usuário airflow:

Terminal input

```
su -c "psql -U postgres -c \"alter user airflow with password 'airflow'\"" -
postgres;
```

3. Crie o banco de dados airflow:

Terminal input

```
su -c 'createdb -O airflow airflow' - postgres;
```

Criação do Banco de Metadados do Apache Druid

1. Crie o usuário druid:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
su -c 'createuser -s -d -l druid' - postgres;
```

2. Defina a senha do usuário druid:

Terminal input

```
su -c "psql -U postgres -c \"alter user druid with password 'druid'\"" -
postgres;
```

3. Crie o banco de dados druid:

Terminal input

```
su -c 'createdb -O druid druid' - postgres;
```

Criação do Banco de Metadados do Apache Hive

1. Crie o usuário hive:

Terminal input

```
su -c 'createuser -s -d -l hive' - postgres;
```

2. Defina a senha do usuário hive:

Terminal input

```
su -c "psql -U postgres -c \"alter user hive with password 'hive'\"" -
postgres;
```

3. Crie o banco de dados hive:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
su -c 'createdb -O hive hive' - postgres;
```

Criação do Banco de Metadados do Jupyterhub

1. Crie o usuário Jupyterhub:

Terminal input

```
su -c 'createuser -s -d -l jupyterhub' - postgres;
```

2. Defina a senha do usuário Jupyterhub:

Terminal input

```
su -c "psql -U postgres -c \"alter user jupyterhub with password
'jupyterhub'\"" - postgres;
```

3. Crie o banco de dados Jupyterhub:

Terminal input

```
su -c 'createdb -O jupyterhub jupyterhub' - postgres;
```

Criação do Banco de Metadados do Openmetadata

1. Crie o usuário Openmetadata:

Terminal input

```
su -c 'createuser -s -d -l openmetadata' - postgres;
```

2. Defina a senha do usuário Openmetadata:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
su -c "psql -U postgres -c \"alter user openmetadata with password
'openmetadata'\"" - postgres;
```

3. Crie o banco de dados Openmetadata:

Terminal input

```
su -c 'createdb -O openmetadata openmetadata' - postgres;
```

Criação do Banco de Metadados do Apache Ranger

1. Crie o usuário ranger:

Terminal input

```
su -c 'createuser -s -d -l ranger' - postgres;
```

2. Defina a senha do usuário ranger:

Terminal input

```
su -c "psql -U postgres -c \"alter user ranger with password 'ranger'\"" -
postgres;
```

3. Crie o banco de dados ranger:

Terminal input

```
su -c 'createdb -O ranger ranger' - postgres;
```

Criação do Banco de Metadados do Ranger KMS

1. Crie o usuário ranger:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
su -c 'createuser -s -d -l rangerkms' - postgres;
```

2. Defina a senha do usuário ranger:

Terminal input

```
su -c "psql -U postgres -c \"alter user rangerkms with password
'rangerkms'\"" - postgres;
```

3. Crie o banco de dados ranger:

Terminal input

```
su -c 'createdb -O rangerkms rangerkms' - postgres;
```

Criação do Banco de Metadados do Apache Superset

1. Crie o usuário superset:

Terminal input

```
su -c 'createuser -s -d -l superset' - postgres;
```

2. Defina a senha do usuário superset:

Terminal input

```
su -c "psql -U postgres -c \"alter user superset with password 'superset'\""
- postgres;
```

3. Crie o banco de dados superset:

Terminal input

```
su -c 'createdb -O superset superset' - postgres;
```

© Copyright Tecnisys 2026. All rights reserved.

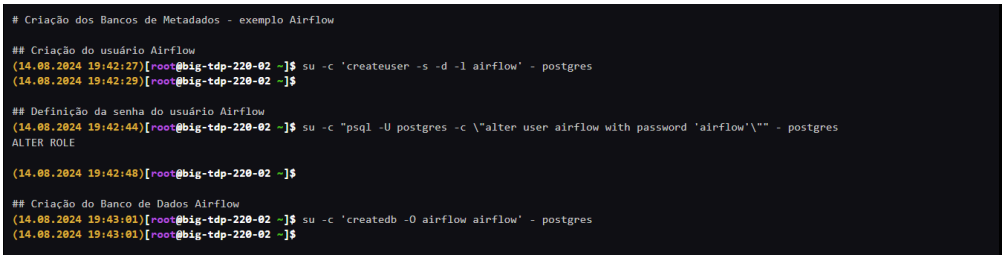


Figura 7 - Criação do metadados

Configuração da Instância dos Bancos de Metadados

A seguir, fornecemos orientações e exemplos de código para a configuração de regras de acesso e propriedades do PostgreSQL (SGBD padrão do Ambari Server). Caso tenha escolhido um SGBD diferente para o seu ambiente, consulte a documentação oficial do mesmo.

> **NOTE**
>
> Verifique, e se necessário altere, a versão majoritária do PostgreSQL presente nos caminhos dos arquivos de configuração.

1. Adicione os demais bancos de dados à regra de acesso do Ambari :

Terminal input

```
sed -i
's/ambari,mapred/ambari,druid,jupyterhub,openmetadata,airflow,ranger,rangerkms,
/var/lib/pgsql/14/data/pg_hba.conf
```

2. Configure a instância PostgreSQL para receber conexões de todas as interfaces de rede:

Terminal input

```
sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g"
/var/lib/pgsql/14/data/postgresql.conf
```

3. Incremente o número máximo de conexões suportadas pela instância PostgreSQL:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
sed -i "s/max_connections = 100/max_connections = 500/g"
/var/lib/pgsql/14/data/postgresql.conf
```

4. Reinicie o serviço do PostgreSQL para que todas as alterações sejam efetivadas:

Terminal input

```
systemctl restart postgresql-14
```

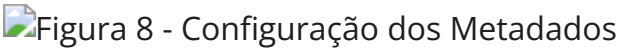


Figura 8 - Configuração dos Metadados

> **WARNING**
>
> Avalie os impactos de tais configurações na segurança e desempenho do seu ambiente. Caso necessite de apoio, Entre em contato, será um prazer auxiliá-lo.

Footnotes

1. Máquinas do tipo *Utility* são normalmente utilizadas para tarefas auxiliares, como o gerenciamento do cluster, enquanto máquinas do tipo *Edge* são dedicadas para interfaces gráficas ou componentes utilizados por usuários na "borda" do ambiente. ↩

© Copyright Tecnisys 2026. All rights reserved.

Criação do Cluster e Instalação dos Componentes

Após a instalação do serviço do Apache Ambari, o próximo passo é a criação do *Cluster de Big Data*, incluindo a instalação dos demais serviços/componentes desejados.

Vamos Começar!

1. Utilize um browser para acessar a interface web do Ambari disponível no IP/hostname da máquina do Ambari Server, porta 8080. Por exemplo:

```
http://192.168.56.100:8080
```

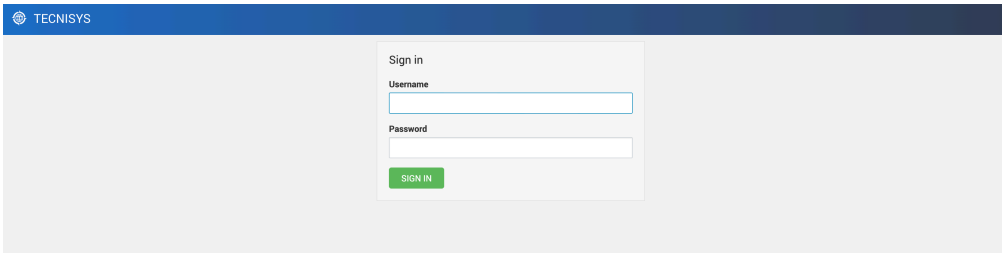


Figura 1 - Página de login do Ambari

> **NOTE**
>
> Por padrão, o usuário / senha de acesso são, respectivamente, *admin / admin*.

Instruções

1. No primeiro acesso será exibida uma página inicial de boas-vindas. Para iniciar o processo de implantação do *Cluster*, clique no botão LAUNCH INSTALL WIZARD:

© Copyright Tecnisys 2026. All rights reserved.

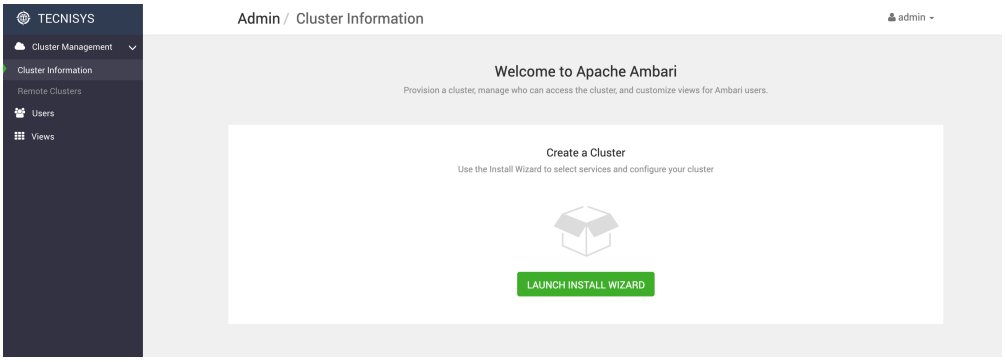


Figura 2 - Página de Boas Vindas do Ambari

2. Informe um nome para o Cluster e clique em NEXT:

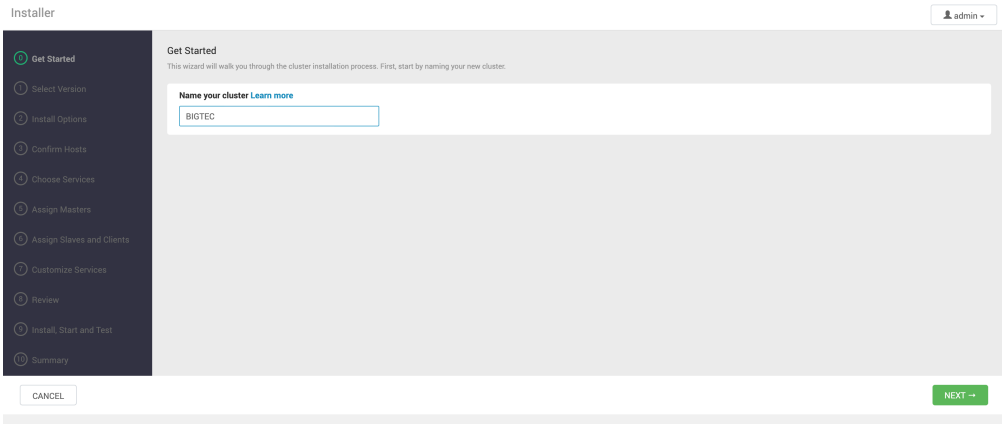


Figura 3 - Nome do Cluster

Seleção da Versão

1. Selecione a versão do TDP desejada:

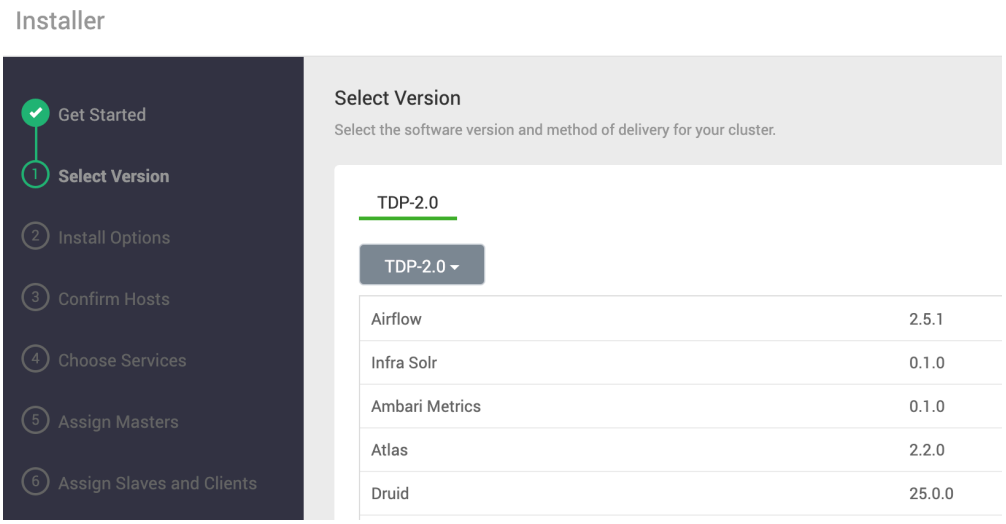


Figura 4 - Versão do TDP

© Copyright Tecnisys 2026. All rights reserved.

2. Selecione o tipo do repositório de pacotes (Public ou Local) e informe a URL para *Components* (TDP-3.0) e *Utils* (TDP-UTILS-3.0):

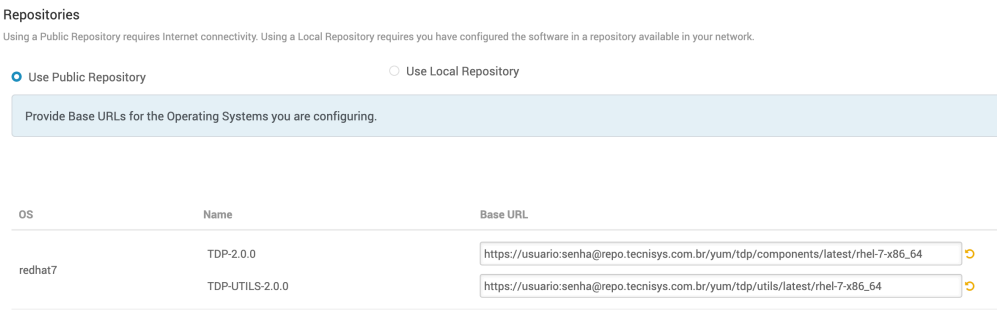


Figura 5 - Repositórios de pacotes

3. Na sequência, clique em NEXT.

> **IMPORTANT**
>
> Caso opte pela utilização do Repositório Público de Pacotes da Tecnisys, as credenciais de acesso (usuário e senha) devem ser informadas diretamente na URL, conforme demonstrado na imagem acima.

Opções de Instalação

1. Em Target Hosts, informe o *Fully Qualified Domain Name* (FQDN) dos hosts (máquinas) que irão compor o *Cluster*.

   O Ambari Server precisa ter acesso às máquinas informadas. Certifique-se de que a resolução do FQDN das máquinas ocorra corretamente, seja através de um Servidor de DNS (recomendado) ou localmente (arquivo `/etc/hosts`).

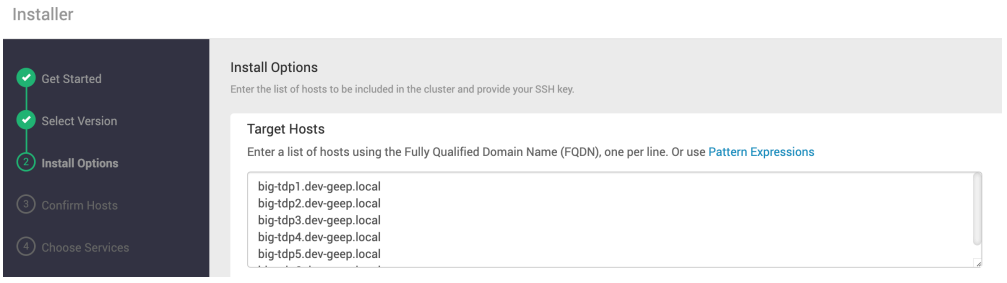


Figura 6 - Informação dos hosts do Cluster

> **TIP**

© Copyright Tecnisys 2026. All rights reserved.

> Em Target Hosts, é possível informar as máquinas usando Expressões (*Pattern Expressions*). O exemplo apresentado na figura acima ficaria assim: `big-tdp[1-7].dev-geep.local`.

2. Em *Hosts Registration Information*, selecione como as máquinas do *Cluster* serão registradas.

   - Caso opte por fornecer a chave SSH privada da máquina do Ambari Server para o registro automático das máquinas do *Cluster*, cole o seu conteúdo no campo de texto abaixo ou faça o *upload* de seu arquivo. Na sequência, confirme o usuário e a porta SSH a serem utilizados. Além disso, certifique-se de que a Relação de Confiança (a troca das chaves SSH) tenha sido realizada corretamente, sendo possível, a partir da máquina do Ambari Server, acessar todas as máquinas via SSH sem a informação da senha do usuário do *daemon* do Ambari Server (por padrão, root).

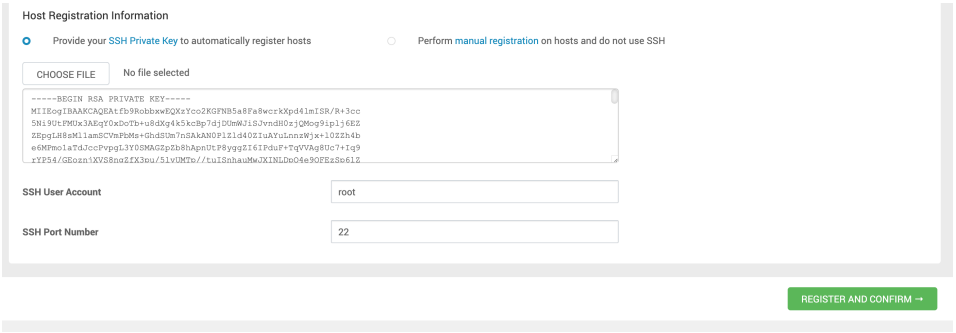


Figura 7 - Registro dos hosts

   - Caso opte por realizar o registro manual das máquinas, faça você mesmo a instalação do Ambari Agent em todas as máquinas antes de prosseguir.

> **TIP**
>
> A chave SSH privada do tipo RSA pode ser obtida executando o seguinte comando:
>
> **Terminal input**
>
> ```
> cat ~/.ssh/id_rsa
> ```

© Copyright Tecnisys 2026. All rights reserved.



> **NOTE**
>
> Para instalar manualmente o Ambari Agent:
>
> **Terminal input**
>
> ```
> yum install ambari-agent
> ```

3. Na sequência, clique em REGISTER AND CONFIRM.

Configuração da Relação de Confiança

1. Na máquina do Ambari Server, gere uma chave SSH privada:

   Terminal input

   ```
   ssh-keygen
   ```

2. Copie a chave SSH para *TODAS* as máquinas do *Cluster*. Por exemplo:

   Terminal input

   ```
   ssh-copy-id tdp-mn01.tecnisys.com.br
   ```

3. Teste o acesso via SSH à *TODAS* as máquinas do *Cluster* sem a informação da senha do usuário. Por exemplo:

   Terminal input

   ```
   ssh root@tdp-mn01.tecnisys.com.br
   ```

Confirmação dos Hosts

Após a instalação do Ambari Agent em todas as máquinas informadas na etapa anterior, o Ambari realiza uma série de verificações para garantir que os pré-requisitos foram atendidos (JDK, Firewall, THP, entre outros).

© Copyright Tecnisys 2026. All rights reserved.

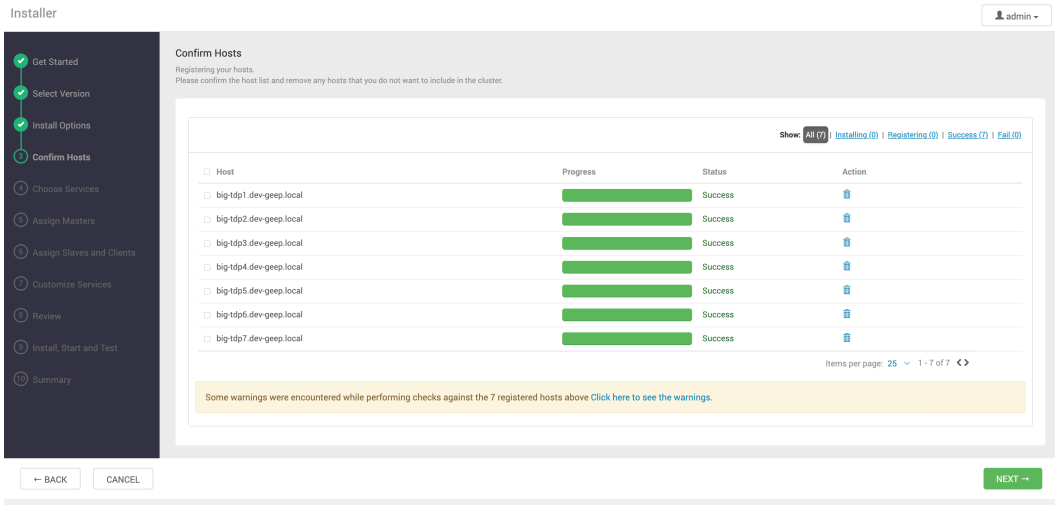


Figura 8 - Confirmação dos hosts

Eventuais erros precisam ser corrigidos e a verificação reexecutada para prosseguir.

Clique em NEXT para avançar.

> **NOTE**
>
> Alertas do tipo *Package Issues*, referentes aos pacotes do PostgreSQL já instalados, podem ser desconsiderados.

Seleção dos Serviços

1. Selecione o serviço responsável pela camada de armazenamento do Cluster.

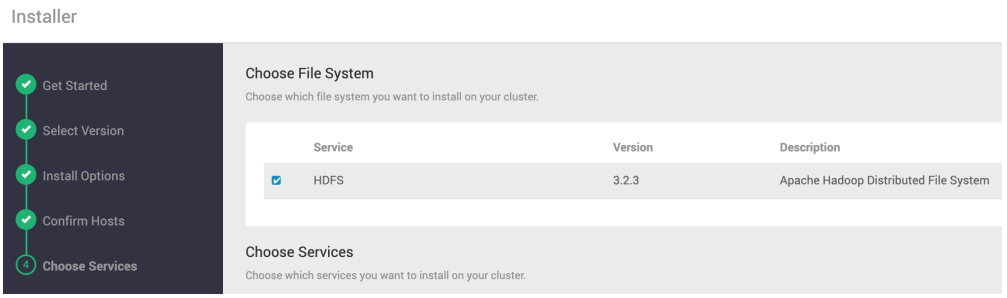


Figura 9 - Seleção do serviço da camada de armazenamento

2. Selecione os demais serviços do Cluster.

© Copyright Tecnisys 2026. All rights reserved.

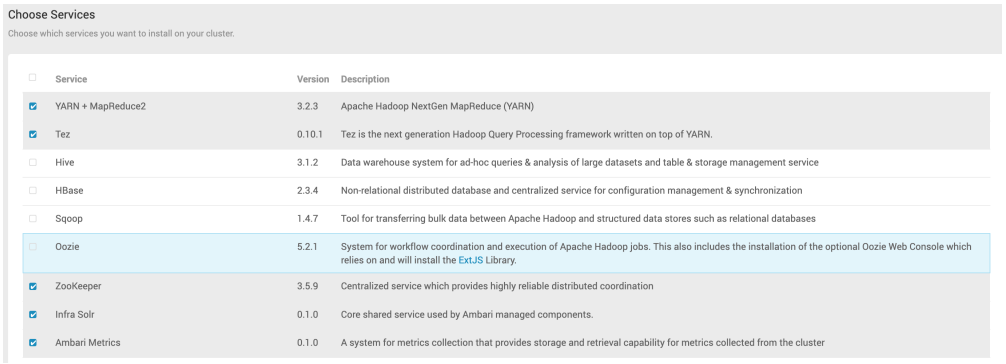


Figura 10 - Seleção dos demais serviços do Cluster

> **TIP**
>
> Recomendamos, inicialmente, a seleção dos serviços básicos, como YARN + MapReduce2, Tez, Zookeeper, Infra Solr e Ambari Metrics. Os demais serviços, caso necessário, podem ser adicionados após a criação do Cluster. Assim é mais fácil lidar com possíveis problemas na instalação dos componentes.

> **NOTE**
>
> O Cluster requer determinados serviços para operar plenamente, como por exemplo, o Apache Ranger para a camada de segurança e o Apache Atlas para a camada de governança de dados. Logo, o Ambari apresentará alertas caso alguma funcionalidade venha a ser limitada pela não instalação de um serviço específico. Ignore o alerta (clique no botão PROCEED ANYWAY) caso o serviço em questão venha a ser instalado futuramente, ou caso esteja ciente de tal limitação.

3. Na sequência, clique em NEXT.

Atribuição dos Componentes Masters

1. Indique a máquina de cada um dos componentes Masters (em geral, componentes de gerenciamento e coordenação) dos serviços selecionados. Observe que à direita da página é apresentada a organização dos componentes por máquina.

© Copyright Tecnisys 2026. All rights reserved.

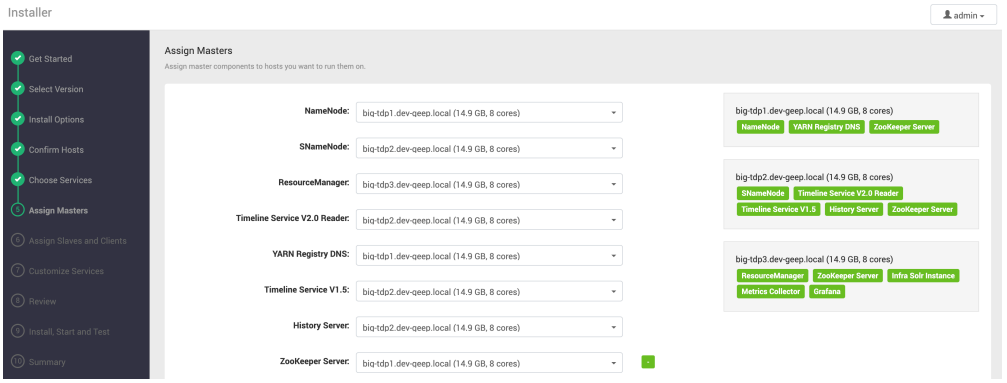


Figura 11 - Atribuição dos componentes Masters

> **NOTE**
>
> A organização deve ser feita considerando as necessidades de cada componente e os recursos disponíveis em cada máquina. Algumas recomendações podem ser observadas:
>
> - Evite instalar na máquina do Ambari Server serviços que não sejam de *Edge* ou *Gateway*. Se possível, deixe uma máquina dedicada para o Ambari Server.
> - Componentes responsáveis pela alta disponibilidade de serviços devem ser instalados em máquinas distintas. Por exemplo, NameNode e Secondary NameNode (SNameNode).
> - Instale o Zookeper em número ímpar de máquinas, maior que um (01). Ou seja, inicialmente, em pelo menos 3 máquinas.

2. Na sequência, clique em NEXT.

Atribuição dos Componentes Slaves e Clients

1. Indique em quais máquinas serão instalados os componentes Slaves (em geral, componentes de armazenamento e processamento) e Clients.

© Copyright Tecnisys 2026. All rights reserved.

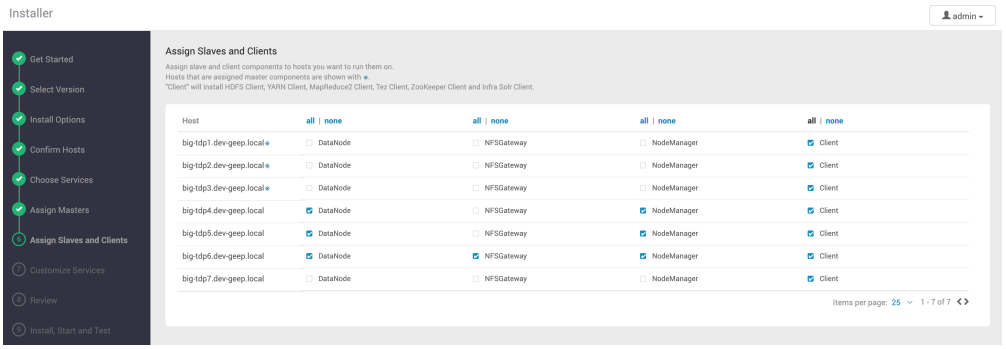


Figura 12 - Atribuição dos componentes Slaves e Clients

> **NOTE**
>
> Sempre que possível, evite instalar componentes Slaves em máquinas de componentes Masters.

2. Na sequência, clique em NEXT.

Customização dos Serviços

Nessa etapa devem ser definidas as credenciais de acesso, dados de conexão a bancos de dados, diretórios, usuários, entre outras informações próprias de cada serviço e necessárias para a instalação.

Resolva as pendências de todas as seções dessa etapa e clique em NEXT para avançar.

Credenciais

Ilustrando essa seção, temos o Grafana, componente do Ambari Metrics, que requer a definição do usuário e senha de administração da ferramenta:

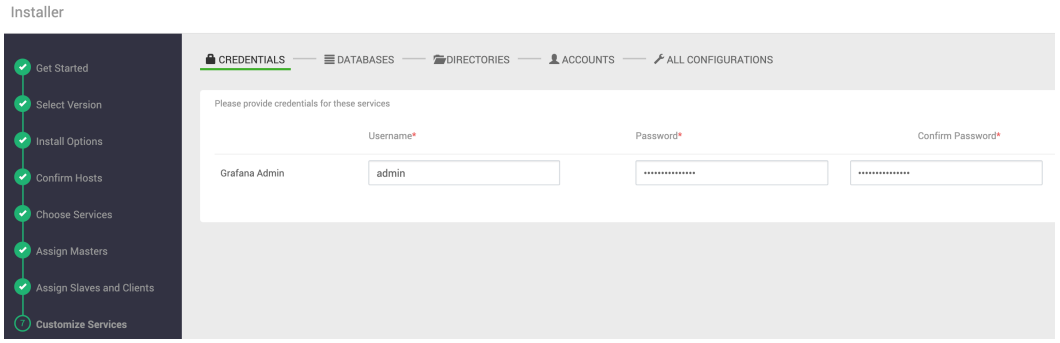


Figura 13 - Definição das credenciais de administração do Grafana

Bancos de Dados

© Copyright Tecnisys 2026. All rights reserved.

Ilustrando essa seção, temos o Hive, o qual requer um banco de dados para persistência de metadados. Nesse exemplo, informamos os dados de conexão para uma instância PostgreSQL existente:

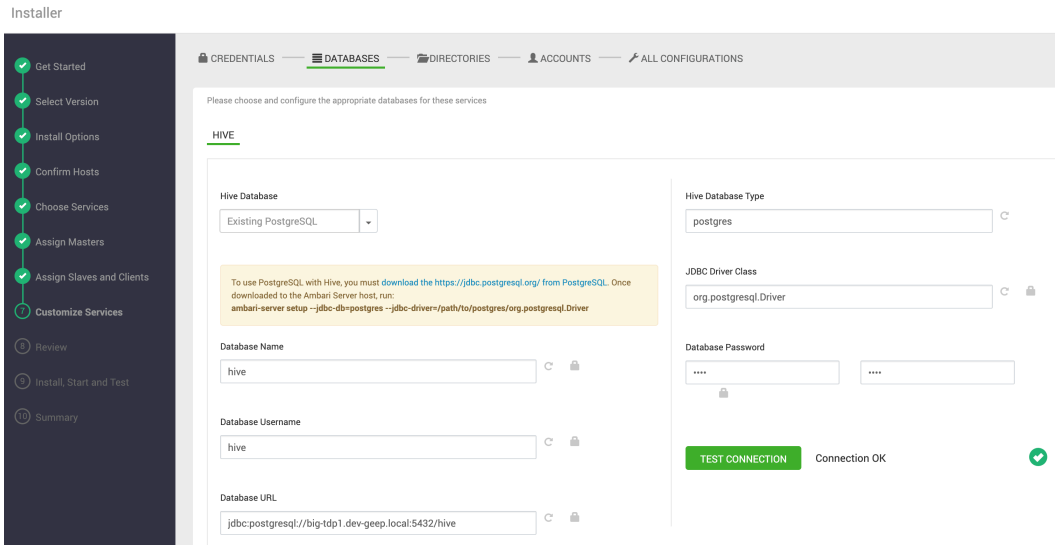


Figura 14 - Definição dos dados de conexão do banco de dados do Hive

> **NOTE**
>
> Clique no botão TEST CONNECTION para testar a conexão com o banco de dados informado.

Diretórios

Nesta seção é possível customizar os diretórios dos serviços, como por exemplo, os diretórios de dados dos DataNodes, os diretórios da namespace do NameNode, diretórios de log, etc.

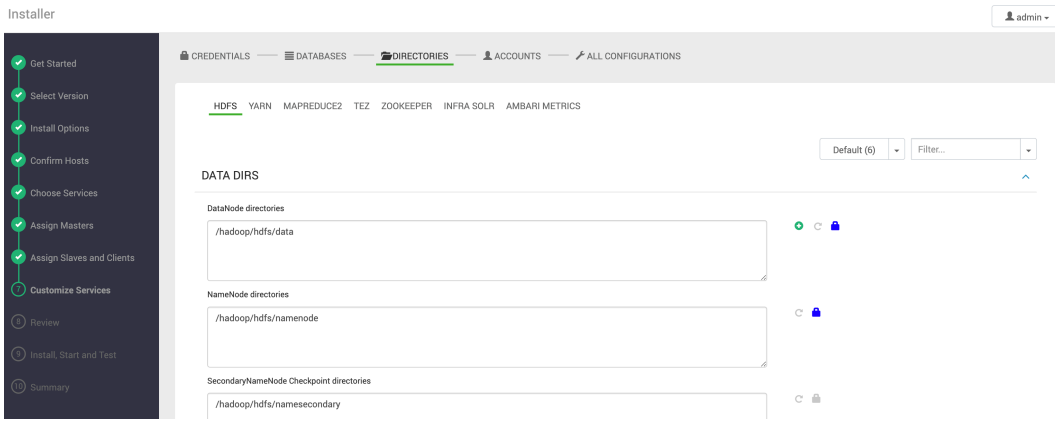


Figura 15 - Definição dos diretórios dos serviços

© Copyright Tecnisys 2026. All rights reserved.



> **WARNING**
>
> Se possível, use dispositivos de armazenamento (discos, SSD, entre outros), volumes e diretórios exclusivos para os arquivos dos DataNodes, NameNodes, JournalNodes, NodeManagers, Timeline Services e Zookeeper.

Usuários dos Serviços

Nesta seção é possível customizar os usuários de sistema operacional que serão criados para cada serviço.

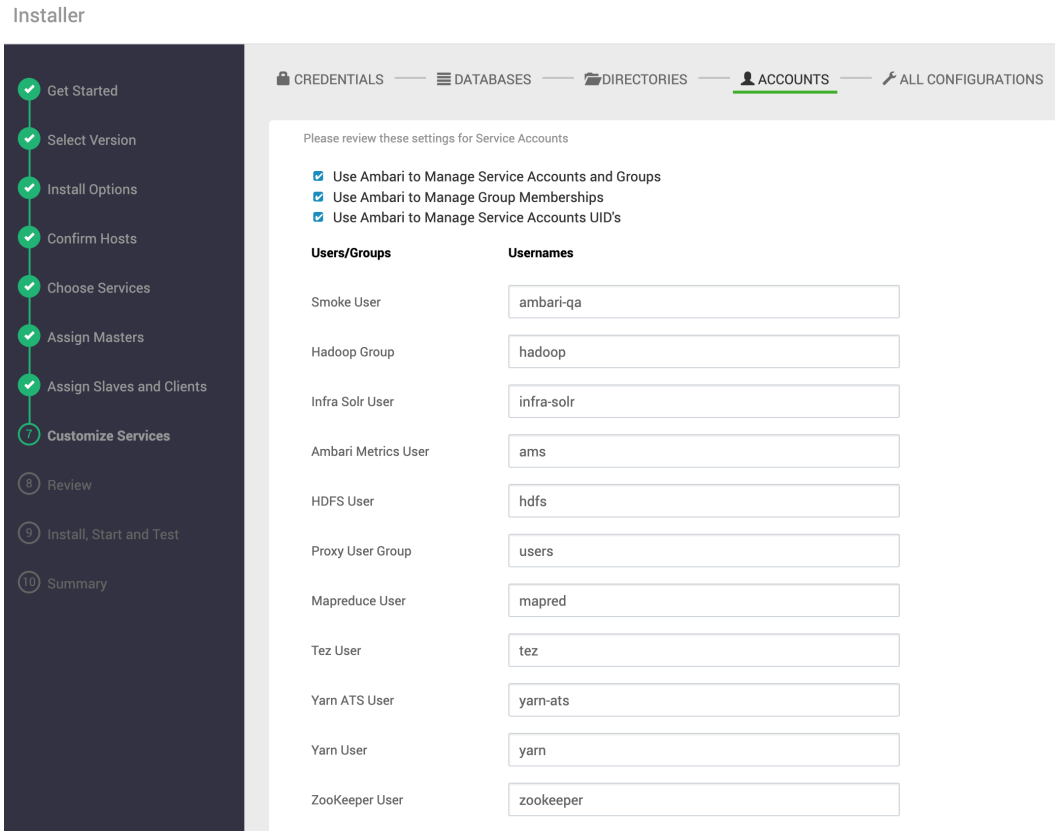


Figura 16 - Definição dos usuários dos serviços

Todas as Configurações

Esta última seção dá acesso a todas as configurações dos serviços a serem instalados. Aproveite para conferir e ajustar o que for necessário.

© Copyright Tecnisys 2026. All rights reserved.

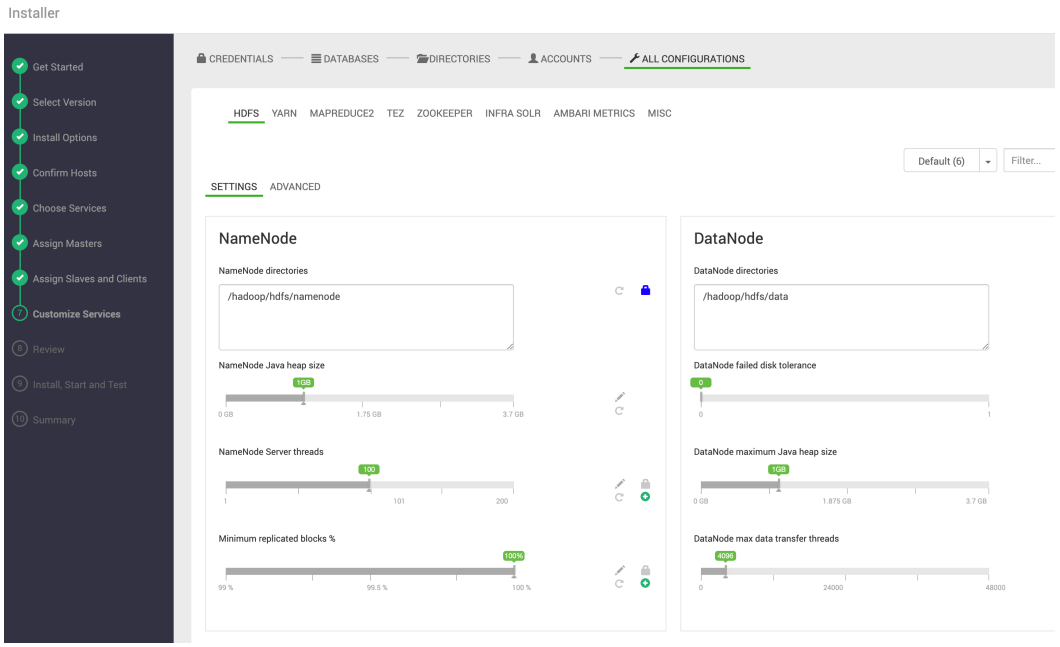


Figura 17 - Todas as configurações dos serviços

Caso tenha esquecido alguma configuração, não se preocupe. Após a instalação, todas essas configurações também estarão disponíveis para alteração via Ambari.

Revisão das Configurações

Nessa etapa, a última antes da criação do Cluster, é apresentada uma revisão das configurações definidas. Verifique cuidadosamente todas as informações e, sendo necessário alterar alguma configuração, utilize a área de navegação lateral esquerda para retornar na etapa desejada.

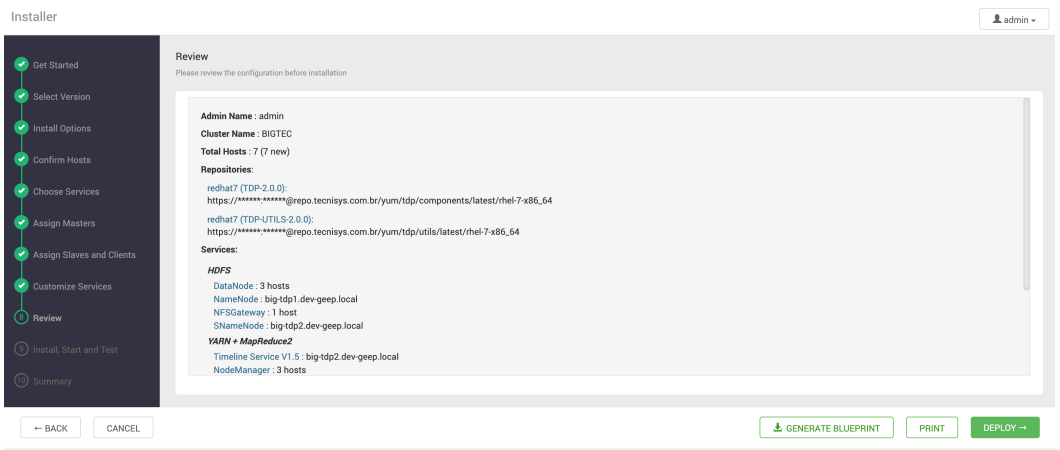


Figura 18 - Revisão da configuração antes da instalação

TIP

© Copyright Tecnisys 2026. All rights reserved.



> Utilize o botão PRINT para gerar um relatório da instalação e o botão GENERATE BLUEPRINT para gerar um arquivo XML com todas as configurações definidas e que, futuramente, pode ser utilizado para recriar o Cluster via Ambari REST API.

Para iniciar a implantação do Cluster, clique no botão DEPLOY.

Instalação, Iniciação e Teste dos Serviços

Nessa etapa, os serviços serão instalados, iniciados e testados, respeitando as dependências e integrações de cada um.

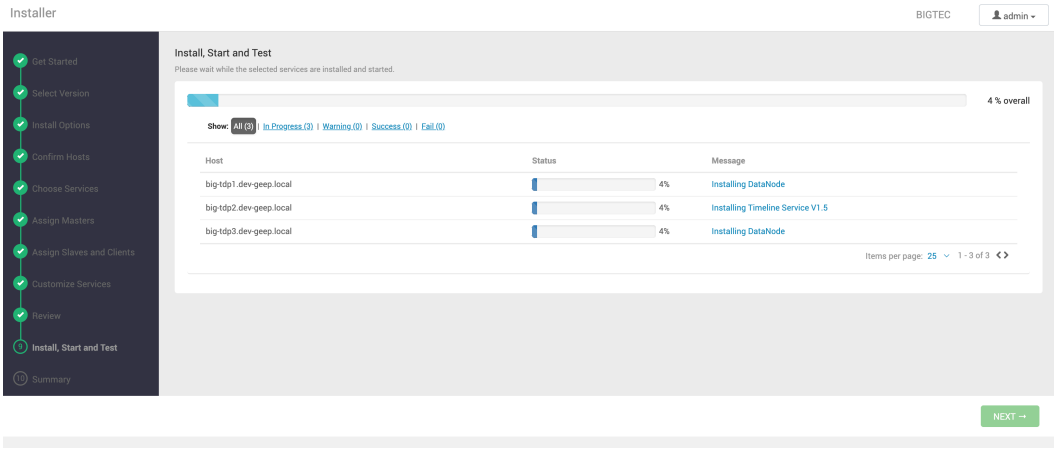


Figura 19 - Instalação dos serviços

> **NOTE**
>
> Clique no link do texto da coluna Message para visualizar as tarefas programadas para cada máquina.

Na ocorrência de falhas, o Ambari pode interromper a implantação, sendo possível retomá-la após a correção do problema clicando no botão RETRY.

No entanto, de acordo com o progresso já realizado, o Ambari pode concluir a implantação e disponibilizar o Cluster como ele está, mesmo que nem todos os componentes de um determinado serviço tenham sido instalados, iniciados ou testados com sucesso. Nesse caso, após criado o Cluster, é possível, pelo próprio Ambari, alterar as configurações ou remover e instalar novamente apenas o serviço problemático.

Finalizada a implantação, clique em NEXT.

© Copyright Tecnisys 2026. All rights reserved.

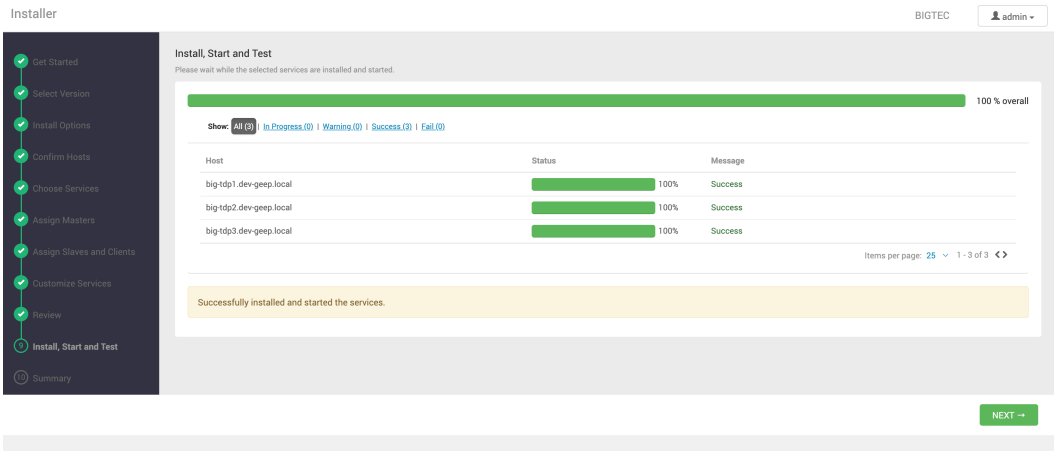


Figura 20 - Instalação, Iniciação e Testes finalizados

Sumário

Na última etapa do processo é apresentado um resumo da implementação.

Clique no botão COMPLETE para finalizar a operação e acessar a área de administração do *Cluster* criado.

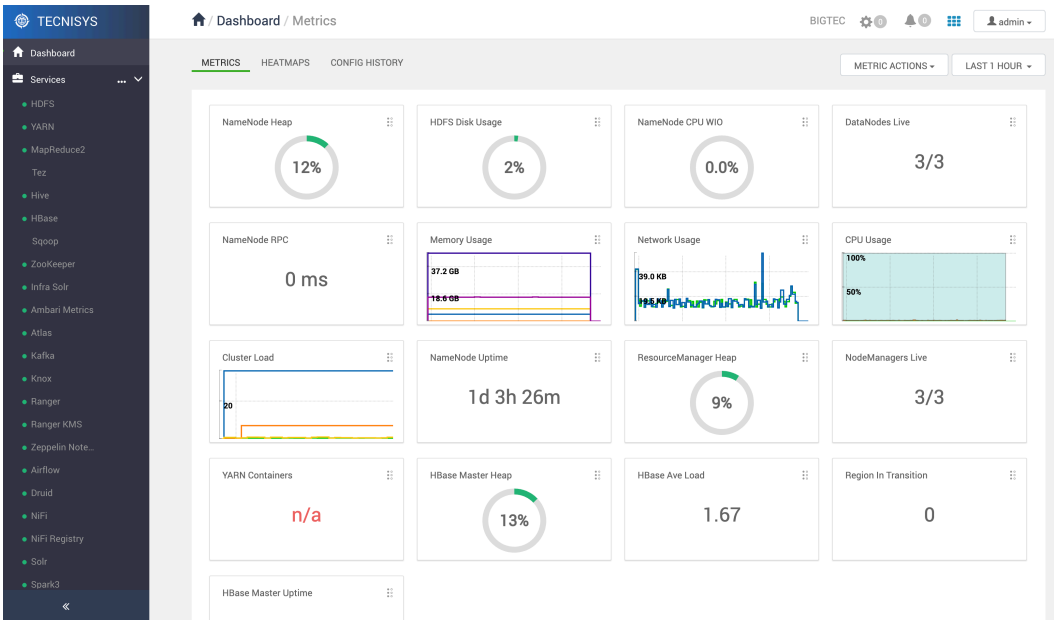


Figura 21 - Área de administração do Cluster

© Copyright Tecnisys 2026. All rights reserved.

Instalação do tdpctl

O tdpctl é uma Interface de Linha de Comando (CLI), desenvolvida pela Tecnisys, para automação de tarefas e instalação de extensões no TDP.

Pré-Requisitos

- Sistema operacional Enterprise Linux, tais como Rocky Linux, AlmaLinux e Red Hat, versão 8 ou superior.
- Privilégio de superusuário (root) na máquina para instalação.

> **NOTE**
>
> Os seguintes RPMs precisam estar disponíveis para instalação via gerenciador de pacotes (dnf): python3-devel, krb5-server, krb5-libs, krb5-workstation, krb5-devel e gcc.

Download

O tdpctl é fornecido no formato tarball para sistemas operacionais Enterprise Linux, versão 8 ou superior.

Para baixar a última versão do tdpctl, faça o download do arquivo `tdpctl-[version]-linux-x64.tar.gz` no diretório `/yum/tdp/tdpctl` do Repositório Público de Pacotes da Tecnisys.

Para acessar o repositório, será necessário informar suas credenciais de acesso, definidas no momento do seu cadastro gratuito no site da Tecnisys.

Instalação

Para instalar o tdpctl, siga os passos abaixo:

© Copyright Tecnisys 2026. All rights reserved.

Instruções

1. Descompacte o arquivo tdpctl-versao-linux-x64.tar.gz no diretório `/usr/local`

Terminal input

```
sudo tar -C /usr/local -xzf tdpctl-0.1-alpha-linux-x64.tar.gz
```

2. Execute o script 'setup.sh':

Terminal input

```
cd /usr/local/tdpctl && sudo ./setup.sh
```

> **NOTE**
>
> Após a instalação, o comando `tdpctl` estará disponível a partir de qualquer diretório.



Figura 1 - Setup do TDPCTL

© Copyright Tecnisys 2026. All rights reserved.

Uso do TDPCTL

Instruções

Sintaxe do Comando

Terminal input

```
tdpctl [OPÇÕES] COMANDO [ARGS]...
```

OPÇÕES:

- `--version` => Mostra a versão e sai.
- `--help` => Mostra a ajuda para execução do comando.

COMANDO:

- `addon-install` => Instala uma extensão do TDP.
- `install` => Instala o Apache Ambari para o TDP.

Exemplos:

1. Instalação de uma extensão:

   Terminal input

   ```
   tdpctl addon-install -f nome-do-arquivo-extensão.taf
   ```

2. Detalhes do comando:

   Terminal input

   ```
   tdpctl
   ```

3. Para ajuda:

   Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
tdpctl --help
```

4. Verificar a versão:

Terminal input

```
tdpctl --version
```

5. Ajuda sobre um comando específico:

Terminal input

```
tdpctl addon-install --help
```

> **NOTE**
>
> O arquivo `extensão.taf` é criptografado e só pode ser aberto pelo TDPCTL. A extensão `.taf` é específica para arquivos Tecnisys.

> **NOTE**
>
> Sempre que necessário, consulte a ajuda to tdpctl (opção *--help*) para obter informações detalhadas sobre os comandos e opções disponíveis.

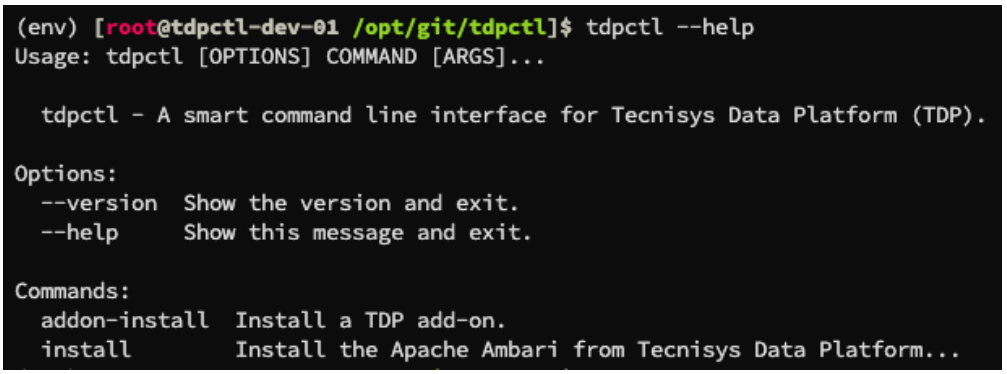


Figura 2 - Uso do TDPCTL

© Copyright Tecnisys 2026. All rights reserved.

ATUALIZAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Fluxograma de Atualização

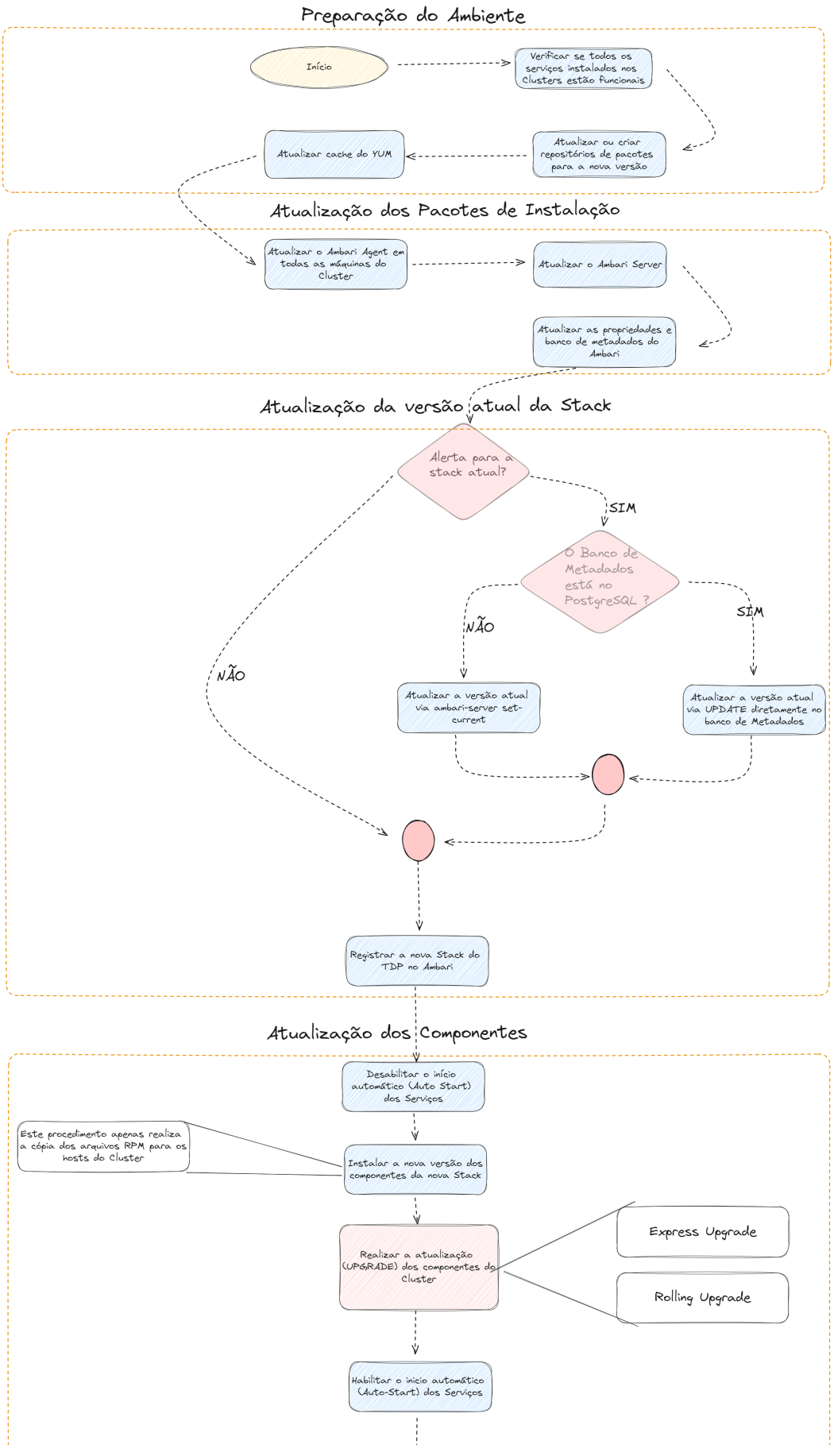
O fluxograma a seguir representa graficamente o processo de atualização de um Cluster TDP.

Resumidamente, o processo de atualização do TDP se divide em três etapas principais:

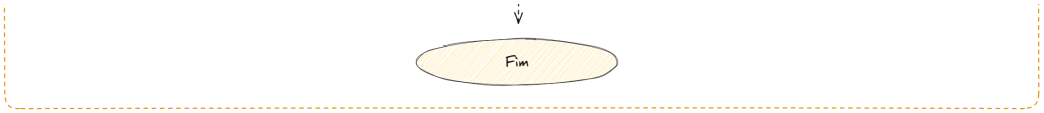
- Atualização do Apache Ambari;
- Registro da nova versão da *stack* do TDP; e
- Atualização dos componentes instalados no *Cluster*.

Nas próximas seções, detalharemos cada uma dessas etapas.

© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.

Pré-Requisitos para Atualização

A seguir apresentamos os pré-requisitos mínimos para a atualização de um *Cluster* TDP, podendo haver outros de acordo com os serviços desejados ou as necessidades técnicas e organizacionais de cada ambiente.

Caso necessite, solicite apoio na Área de Suporte.

Espaço em Disco

A atualização requer a instalação de novos pacotes, os quais demandam de espaço em disco adicional, seja no repositório de pacotes ou nos diretórios das máquinas do *Cluster* destinados à binários, arquivos de configuração e bibliotecas. Para uma atualização segura, sem intercorrências, recomendamos um mínimo de 100 GB de espaço livre em disco.

> **WARNING**
>
> Antes da instalação dos pacotes da nova versão, verifique o espaço em disco disponível, evitando assim a corrupção de arquivos, reinício da operação, entre outros problemas.

Rede de Comunicação

A comunicação entre as máquinas do *Cluster* e o repositório de pacotes deve ser garantida, seja por meio de uma rede local ou pela internet. A velocidade e estabilidade da rede influencia diretamente no tempo de execução e integridade da operação.

Função "Auto Start" Desativada

Antes de iniciar atualização dos componentes é importante desativar a função de Início Automático (*Auto Start*) dos serviços no Ambari. Para isso, siga os passos a seguir:

Instruções

- Selecione no menu lateral do Ambari a opção `Service Auto-Start`.
- Mude a chave para `Disabled` na opção `Auto Start Settings`.

© Copyright Tecnisys 2026. All rights reserved.

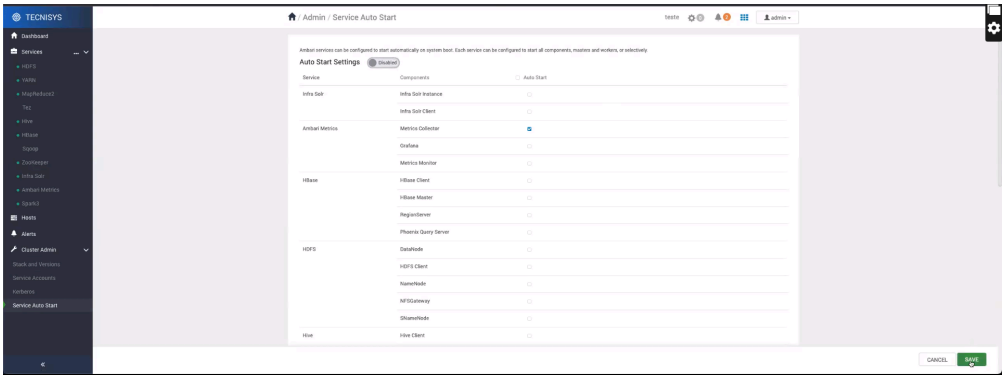


*Figura 5 - Desativando o _Auto Start_*

- Clique em `Save` para salvar a alteração e confirme a operação.

> **NOTE**
>
> Concluída a atualização, a função *Auto Start* pode ser habilitada novamente.

Cargas de Trabalho Finalizadas

Recomendamos que as principais cargas de trabalho sejam finalizadas antes da atualização. Isso evita a interrupção de processos em execução, garantindo a integridade dos dados e a continuidade das operações.

Serviços em Alta Disponibilidade

A atualização do tipo `Rolling Upgrade` é um processo que permite a atualização de um serviço sem interromper o funcionamento do *Cluster*. Para isso, é necessário que os serviços estejam configurados para alta disponibilidade, garantindo a continuidade das operações.

Backup dos Bancos de Metadados

Vários componentes da plataforma TDP utilizam bancos de metadados para armazenar informações sobre configurações, operações, entre outros.

Antes de iniciar a atualização, é recomendado realizar um backup dos bancos de metadados, garantindo assim a possibilidade de restauração em caso de falhas.

Configurações Adicionais no Apache Ambari para Clusters Grandes

© Copyright Tecnisys 2026. All rights reserved.



Em um Cluster de grande porte, com dezenas ou centenas de máquinas, algumas configurações adicionais no Apache Ambari podem ser necessárias para garantir o sucesso da atualização.

Ajuste do Timeout para a Instalação de Pacotes

Em um grande Cluster, a instalação de pacotes através do Ambari pode requerer bastante tempo. Logo, para evitar problemas de *timeout*, aumente o valor do parâmetro `agent.package.install.task.timeout`, localizado no arquivo de configuração `/etc/ambari-server/conf/ambari.properties` da máquina do Ambari Server. Para isso, siga os passos a seguir:

Instruções

1. Abra o arquivo de configurações do Ambari Server com um editor de texto:

Terminal input

```
vim /etc/ambari-server/conf/ambari.properties
```

1.1. Ajuste o valor da propriedade `agent.package.install.task.timeout`. Por exemplo, para 1 hora (3600 segundos):

Terminal input

```
agent.package.install.task.timeout=3600
```

1.2. Salve a alteração e feche o arquivo de configuração do Ambari Server.

© Copyright Tecnisys 2026. All rights reserved.

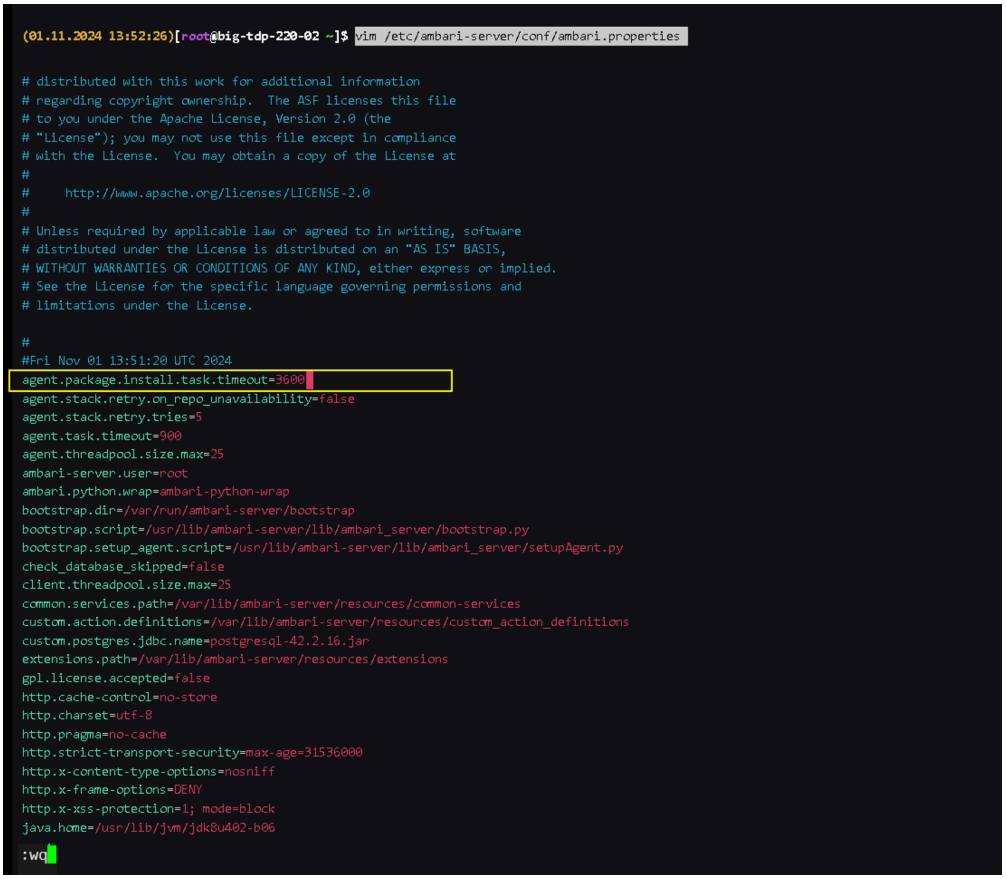


Figura 6 - Ajuste timeout

2. Reinicie o serviço do Ambari Server:

Terminal input

```
ambari-server restart
```

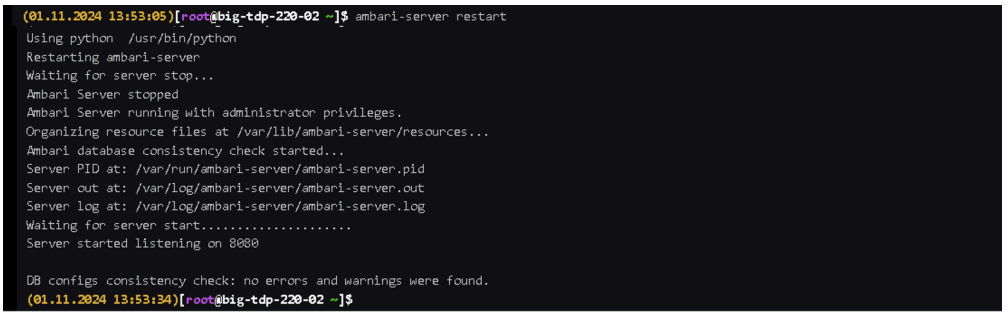


Figura 6 - Ajuste timeout

Ajuste do Tempo de Reinício do NameNode

Em um grande Cluster, o processo de inicialização do NameNode pode demorar significativamente. O tempo de inicialização depende não apenas dos recursos

© Copyright Tecnisys 2026. All rights reserved.

computacionais disponíveis, mas também do volume de dados e parâmetros de rede.

Para garantir que as solicitações do Ambari para iniciar o NameNode não excedam o tempo limite durante uma atualização, configure o parâmetro de tempo limite de reinício do NameNode no Ambari, `upgrade.parameter.nn-restart.timeout`, no arquivo `/etc/ambari-server/conf/ambari.properties` da máquina do Ambari Server.

> **NOTE**
>
> Caso o parâmetro `upgrade.parameter.nn-restart.timeout` não exista no arquivo de configuração, adicione-o.

Instruções

Inicialmente, adicione 10% ao tempo (em segundos) normalmente demandado para reiniciar o NameNode. Embora não haja um método padrão para determinar um valor apropriado, a seguinte orientação pode ser utilizada. Por exemplo, 660 segundos (11 minutos) se o tempo normal de reinicialização for de 600 segundos (10 minutos). Para isso, siga os passos a seguir:

1. Abra o arquivo de configurações do Ambari Server com um editor de texto:

Terminal input

```
vim /etc/ambari-server/conf/ambari.properties
```

1.1 Ajuste o valor da propriedade *upgrade.parameter.nn-restart.timeout*:

Terminal input

```
upgrade.parameter.nn-restart.timeout=660
```

1.2. Salve a alteração e feche o arquivo de configuração do Ambari Server.

2. Reinicie o serviço do Ambari Server:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
ambari-server restart
```

```
(01.11.2024 11:23:30)[root@big-tdp-220-02 ~]$ vim /etc/ambari-server/conf/ambari.properties
server.execution.scheduler.misfire.toleration.minutes=480
server.fqdn.service.url=http://169.254.169.254/latest/meta-data/public-hostname
server.http.session.inactive_timeout=1800
server.jdbc.connection-pool=internal
server.jdbc.database=postgres
server.jdbc.database_name=ambari
server.jdbc.postgres.schema=ambari
server.jdbc.user.name=ambari
server.jdbc.user.passwd=/etc/ambari-server/conf/password.dat
server.os_family=redhat9
server.os_type=rocky linux9
server.persistence.type=local
server.python.log.level=INFO
server.python.log.name=ambari-server-command.log
server.stages.parallel=true
server.task.timeout=1200
server.tmp.dir=/var/lib/ambari-server/data/tmp
server.version.file=/var/lib/ambari-server/resources/version
shared.resources.dir=/usr/lib/ambari-server/lib/ambari_commons/resources
skip.service.checks=false
stack.java.home=/usr/lib/jvm/jdk8u402-b06
stack.jce.name=jce_policy-8.zip
stack.jdk.name=OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
stackadvisor.script=/var/lib/ambari-server/resources/scripts/stack_advisor.py
ulimit.open.files=65536
upgrade.parameter.nn-restart.timeout=660
user.inactivity.timeout.default=0
user.inactivity.timeout.role.readonly.default=0
views.ambari.request.connect.timeout.millis=30000
views.ambari.request.read.timeout.millis=45000
views.http.cache-control=no-store
views.http.charset=utf-8
views.http.pragma=no-cache
views.http.strict-transport-security=max-age=31536000
views.http.x-content-type-options=nosniff
views.http.x-frame-options=SAMEORIGIN
views.http.x-xss-protection=1; mode=block
views.request.connect.timeout.millis=5000
views.request.read.timeout.millis=10000
views.skip.home-directory-check.file-system.list=wasb,adls,adl
webapp.dir=/usr/lib/ambari-server/web

:wq

(01.11.2024 11:24:25)[root@big-tdp-220-02 ~]$ ambari-server restart
Using python  /usr/bin/python
Restarting ambari-server
Waiting for server stop...
Ambari Server stopped
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Ambari database consistency check started...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start.....................
Server started listening on 8080

DB configs consistency check: no errors and warnings were found.
(01.11.2024 11:24:55)[root@big-tdp-220-02 ~]$
```

Figura 6 - Ajuste namenode

© Copyright Tecnisys 2026. All rights reserved.

Atualização do Apache Ambari

> **Pré-Requisitos**
>
> - Criação ou atualização do repositório de pacotes local para atualizações com acesso restrito ou sem acesso à internet.
> - Verificação dos pré-requisitos para a atualização.

Atualização do Apache Ambari

Repositório de Pacotes

Instruções

Execute os procedimentos a seguir, em todas as máquinas do Cluster, para a definição de um novo repositório de pacotes para o Apache Ambari:

- Acesse o diretório em que se encontra o arquivo do repositório de pacotes (*repo file*) do Ambari:

  Terminal input

  ```
  cd /etc/yum.repos.d/
  ```

- Execute o comando abaixo para baixar o arquivo `ambari.repo` e salvá-lo no diretório local /etc/yum.repos.d/, a ser utilizado pelo gerenciador de pacotes YUM.

  Terminal input

  ```
  wget -O /etc/yum.repos.d/ambari.repo
  https://repo.tecnisys.com.br/repository/public/tdp/3.0/el-9-
  ```

© Copyright Tecnisys 2026. All rights reserved.

```
x86_64/ambari.repo
```

Terminal input

```
curl -o /etc/yum.repos.d/ambari.repo
https://repo.tecnisys.com.br/repository/public/tdp/3.0/el-9-
x86_64/ambari.repo
```

> **WARNING**
>
> Os comandos sugeridos abaixo sobrescrevem o arquivo `/etc/yum.repos.d/ambari.repo`, caso ele já exista no diretório. Se necessário, faça antes um *backup* do arquivo com as definições atuais.

- No arquivo ambari.repo, substitua as variáveis USER e PASS, presentes nas URLs das propriedades *baseurl* e *gpgkey*, respectivamente, pelo seu usuário e senha cadastrados no Site da Tecnisys . Caso esteja utilizando um repositório local, ajuste tais URLs.

- Após a definição do repositório de pacotes da nova versão do TDP, recomendamos a atualização do *cache* do yum/dnf:

Terminal input

```
yum clean all; yum makecache timer;
```

Atualização do Ambari Agent

Instruções

Com o novo repositório de pacotes definido, inicie a atualização do Ambari atualizando o componente Ambari Agent.

> **WARNING**
>
> A comunicação entre o Ambari Agent e o Ambari Server é suspensa durante a atualização. Entretanto, os serviços dos demais componentes do *Cluster*

© Copyright Tecnisys 2026. All rights reserved.

continuam em execução.

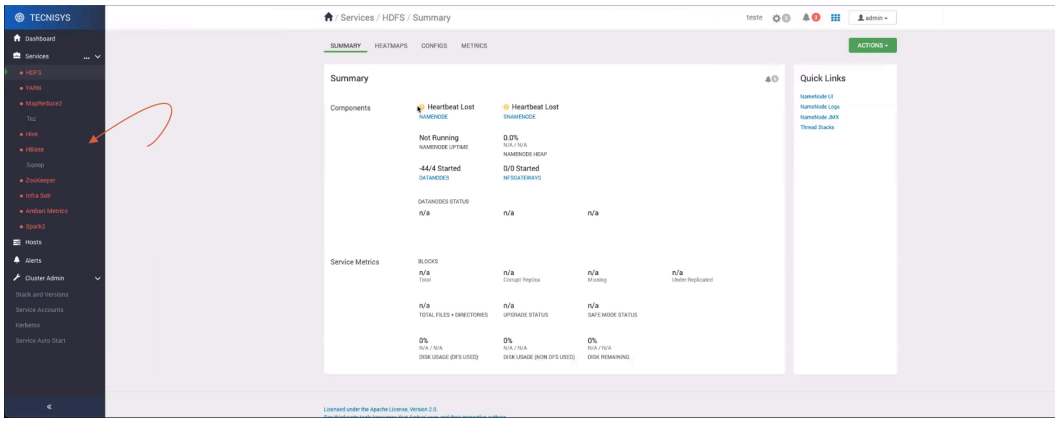


*Figura 1 * Situação dos serviços durante a atualização do Ambari Agent*

Para atualização do Ambari Agent, execute o procedimento a seguir em todas as máquinas do *Cluster*:

- Pare o serviço Ambari Agent:

Terminal input

```
ambari-agent stop
```

- Atualize os binários do Ambari Agent:

Terminal input

```
yum upgrade ambari-agent -y
```

> **NOTE**
>
> Eventuais customizações do arquivo `ambari-agent.ini` podem ser recuperadas no arquivo de backup gerado: `ambari-agent.ini.old`

- Inicie o serviço do Ambari Agent:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
ambari-agent start
```

Atualização do Ambari Server

Instruções

Após a atualização do Ambari Agent em todas as máquinas do *Cluster*, realize a atualização do Ambari Server.

> **WARNING**
>
> Essa operação requer a parada do serviço do Ambari Server.

> **WARNING**
>
> É recomendada a realização de um *backup* do banco de metadados do Ambari antes do início desta operação.

Execute o procedimento a seguir somente na máquina em que está instalado o Ambari Server:

- Pare o serviço do Ambari Server:

Terminal input

```
ambari-server stop
```

- Atualize os binários do Ambari Server:

Terminal input

```
yum upgrade ambari-server -y
```

© Copyright Tecnisys 2026. All rights reserved.

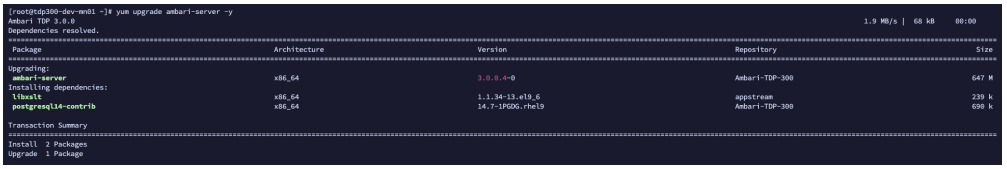


Figura 2 - Atualização dos binários do Ambari Server

- Atualize os metadados do Ambari Server:

Terminal input

```
ambari-server upgrade
```

> **NOTE**
>
> Basicamente, este comando realiza as seguintes operações:
>
> - Ajustes de propriedades do Ambari.
> - Atualização do *schema* do banco de metadados.
> - Verificação de novas versões da *stack* TDP.

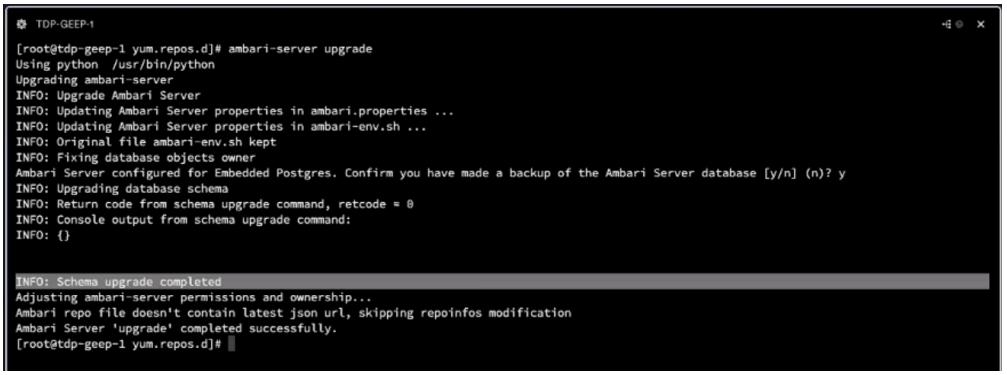


Figura 3 - Ambari Server Upgrade

- Inicie o Ambari Server:

Terminal input

```
ambari-server start
```

- Verifique na interface web a nova versão do Ambari clicando no botão do usuário, localizado no canto superior direito da página, e depois na opção *About*.

© Copyright Tecnisys 2026. All rights reserved.

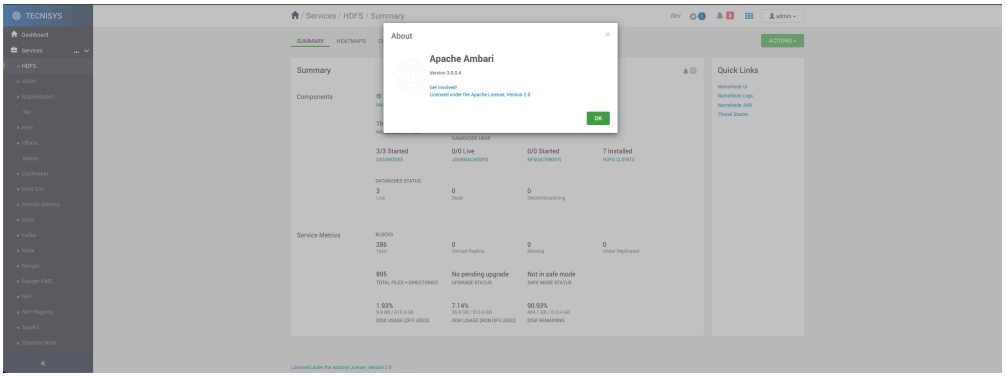


Figura 4 - Versão do Ambari

- Verifique na interface web a nova versão da *stack* do TDP clicando na opção *Stack and Versions* do menu lateral, e depois na guia *VERSIONS*.

> **IMPORTANT**
>
> Caso seja exibido o alerta *´Host component out of sync´* na guia *VERSIONS*, execute o procedimento definido em Atualização da Versão Atual da *Stack* para corrigir esta inconsistência.

© Copyright Tecnisys 2026. All rights reserved.

Registro da Nova Versão

A seguir, são apresentados os procedimentos para registrar a nova versão da *stack* do TDP no Ambari:

- Na interface web do Ambari, acesse a página "Stack and Versions" pelo menu lateral, clique na guia "VERSIONS" e depois clique no botão "MANAGE VERSIONS".

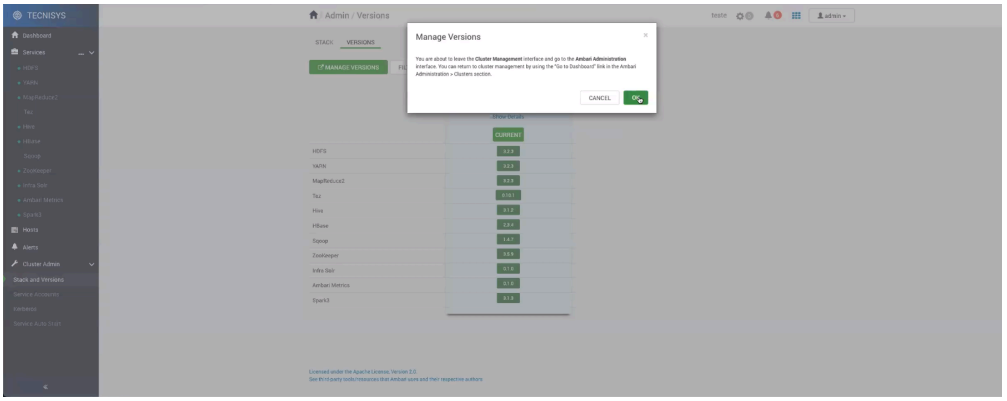


Figura 1 - Redirecionamento para a área de administração de versões

- Confirme o redirecionamento para a área de administração de versões do Ambari.

- Selecione "Register Version".

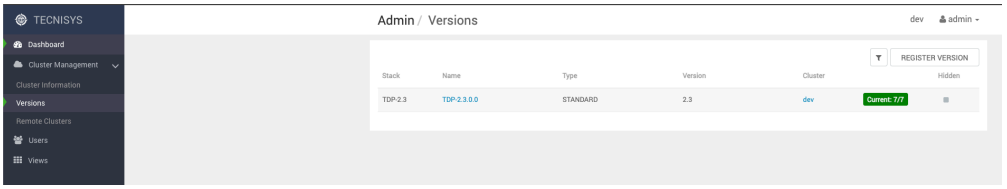


Figura 2 - Área de administração de versões

- No registro da nova versão, informe a *Base URL* dos repositórios de componentes (TDP-3.0) e utilitários (TDP-UTILS-3.0).

© Copyright Tecnisys 2026. All rights reserved.

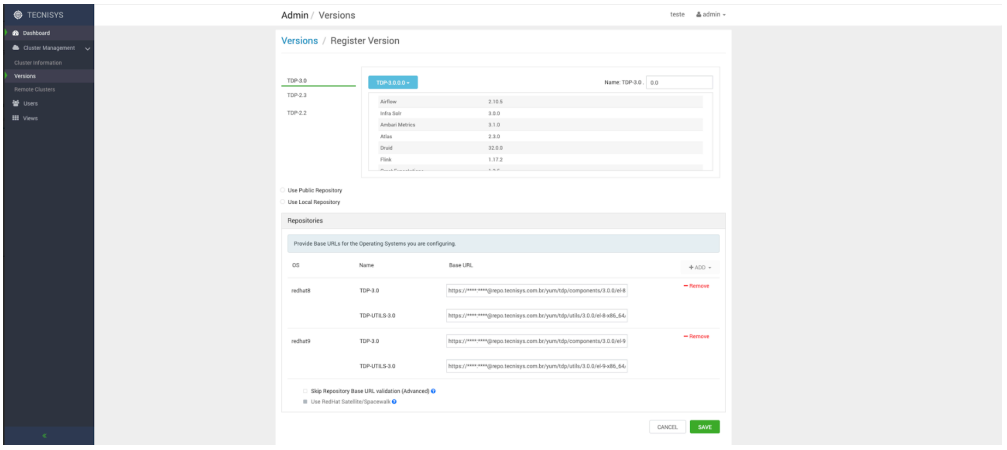


Figura 3 - Registro da nova versão

- Salve as alterações.

- Na sequência, na guia *VERSIONS* já estará disponível a nova versão da *stack* do TDP.

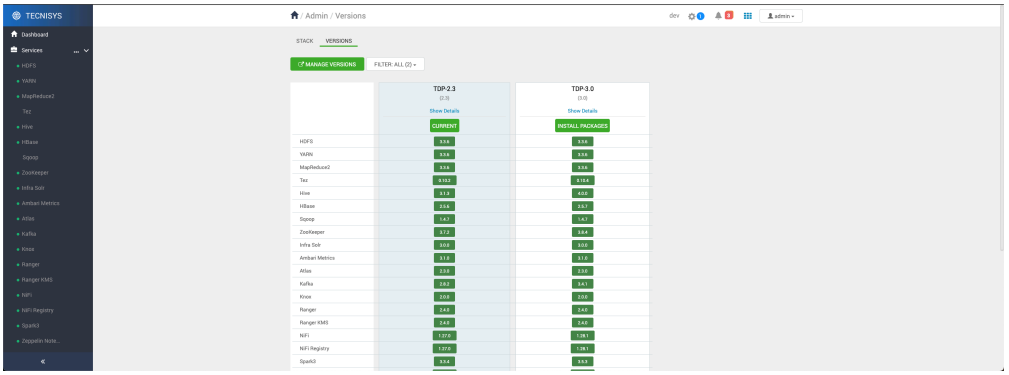


Figura 4 - Nova versão disponível

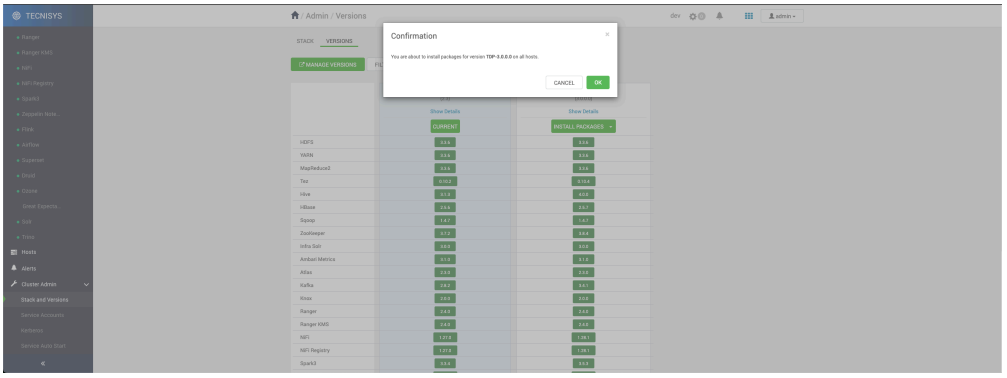
© Copyright Tecnisys 2026. All rights reserved.

Atualização dos Componentes

A atualização dos componentes é realizada através do Ambari. O processo é dividido em duas etapas: instalação dos pacotes disponíveis na nova versão e execução do *upgrade*.

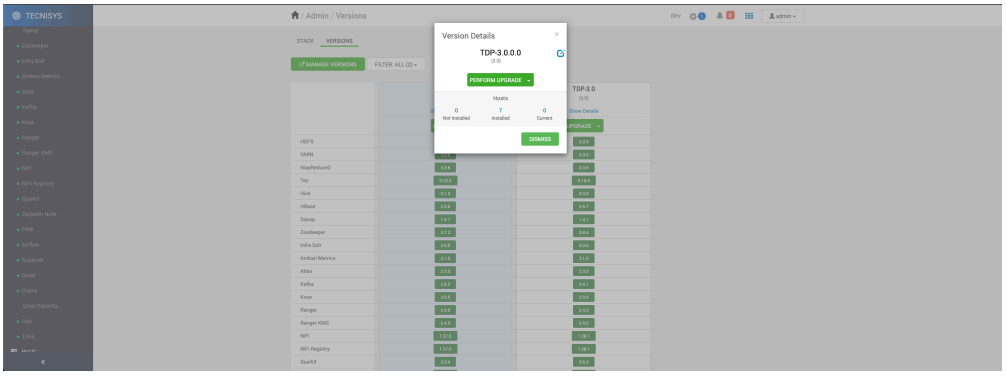
Instalação dos Pacotes

1. No Ambari, na página "Stack and Versions", guia VERSIONS, clique no botão INSTALL PACKAGES da nova versão.



2. Confirme a instalação dos pacotes.

3. Clique no link "Show details" de ambas as versões para visualizar o total de máquinas com pacotes instalados.

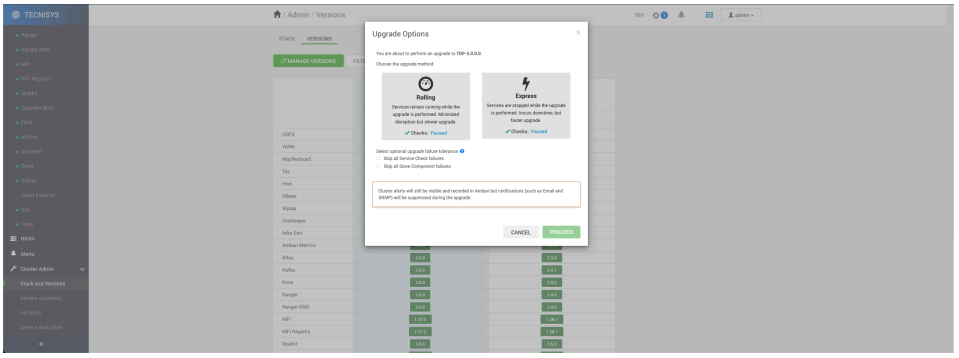


> **NOTE**
>
> Até este momento, mesmo que os pacotes da nova versão já estejam instalados nas máquinas, o *Cluster* ainda opera com os binários dos

© Copyright Tecnisys 2026. All rights reserved.

componentes da versão corrente ("CURRENT"). Nesta etapa, qualquer problema com a atualização se restringirá à instalação dos RPMs.

- A partir deste momento, serão disponibilizadas duas opções para a realização do *Upgrade*:
  - *Express Upgrade*: Essa é a opção mais rápida e todos os serviços do Cluster serão suspensos para a atualização;
  - *Rolling Upgrade*: A atualização é realizada sem a parada total dos serviços do Cluster.



> **IMPORTANT**
>
> Antes de iniciar a atualização, certifique-se que todos os pré-requisitos foram atendidos, os serviços estão funcionais, e a função Auto Start está desativada no Ambari.

Execução do Express Upgrade

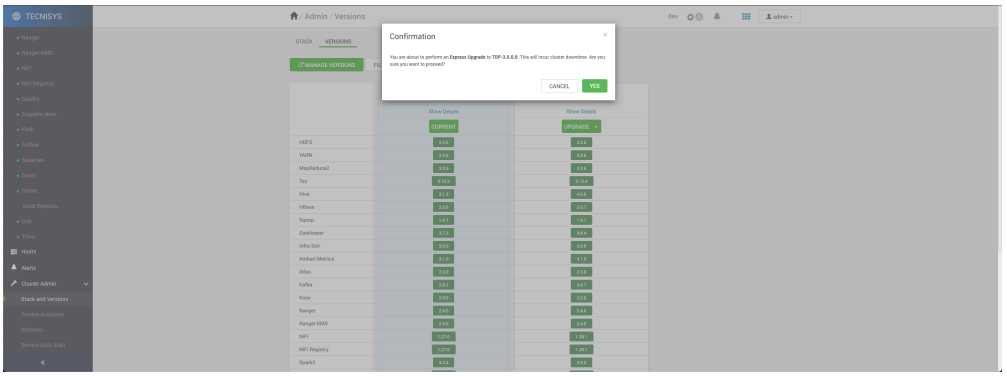
O *Express Upgrade* é uma opção que permite a atualização dos componentes do *Cluster* para uma a nova versão de forma rápida e segura. Contudo, esta opção requer a parada total dos serviços do *Cluster*.

Estando todos os pré-requisitos atendidos, siga os passos a seguir:

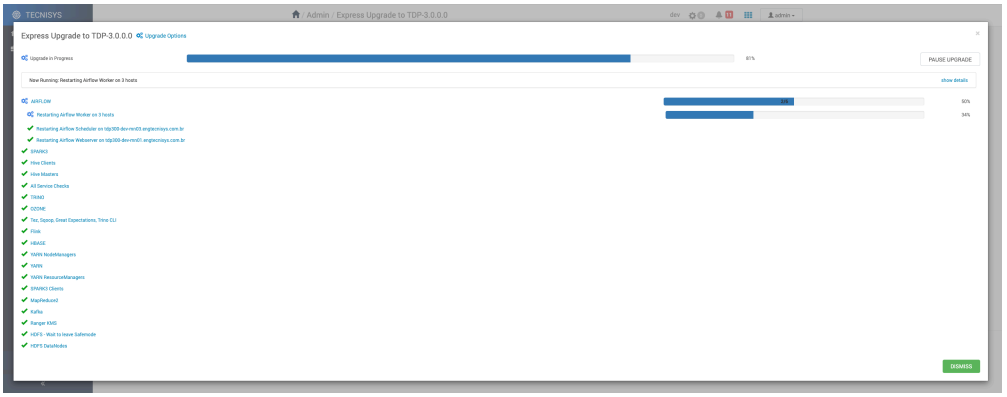
1. Clique na botão UPGRADE da nova versão.
2. Selecione a opção de atualização "Express".
3. Clique em "Proceed" para continuar.

© Copyright Tecnisys 2026. All rights reserved.

4. Confirme a Operação.

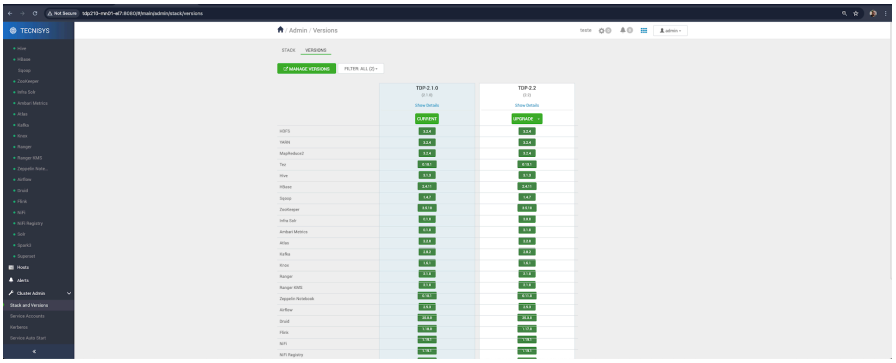


5. Acompanhe o processo de atualização e fique atento às mensagens e solicitações exibidas na tela.



WARNING

Durante o *Express Upgrade*, os serviços ficam indisponíveis. O tempo total de indisponibilidade depende de vários fatores, como a quantidade de máquinas, a quantidade de serviços, velocidade da rede, entre outros.



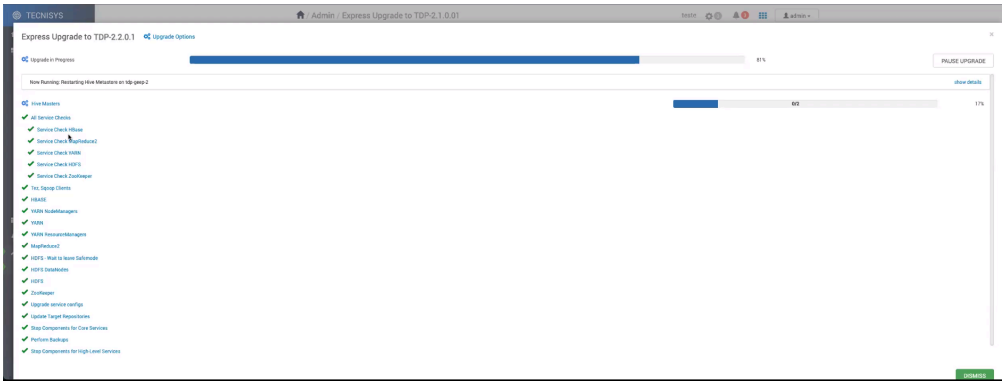
© Copyright Tecnisys 2026. All rights reserved.



> **WARNING**
>
> Caso o serviço do Apache Hive esteja instalado, durante a atualização, será solicitado a realização de um *backup* do banco de dados do Hive Metastore. Esta ação é necessária para viabilizar a recuperação em caso de falha ou o *downgrade* da versão. Na sequência, de volta ao Ambari, confirme a realização do *backup* para prosseguir com a atualização.

> **TIP**
>
> É possível verificar se os componentes atualizados já estão funcionais. Para isto, clique o link "all service checks". Assim é possível identificar e corrigir eventuais problemas antes da etapa de verificação dos serviços (*service check*), realizada ao final da atualização.

6. Acompanhe a verificação de todos os serviços atualizados (*Service Check*).



7. Selecione uma das seguintes opções para concluir o processo de atualização:

- *DOWNGRADE*: Permite reverter o processo de atualização. Todos os componentes voltam para suas versões anteriores.
- *FINALIZE LATER*: Permite finalizar o processo de atualização em outro momento, possibilitando a execução de testes e validações.
- *FINALIZE*: Finaliza o processo de atualização e define a nova versão como "CURRENT".

© Copyright Tecnisys 2026. All rights reserved.



> **WARNING**
>
> Caso o Apache HBase esteja entre os serviços atualizados, ao final do upgrade será solicitada a confirmação da exclusão do snapshot criado durante o processo.

8. Selecione "Finalize" para finalizar o processo de atualização.



9. Clique em "Proceed" para confirmar a finalização do processo de atualização.

Execução do Rolling Upgrade

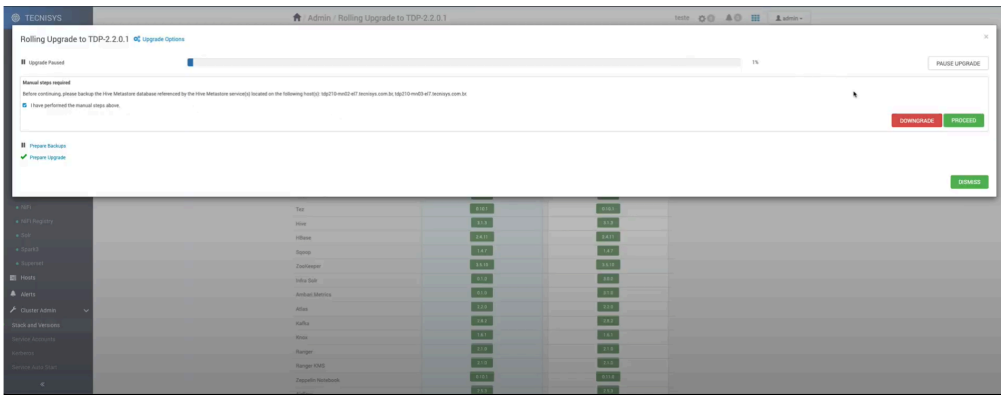
Embora seja uma opção mais demorada, o *Rolling Upgrade* permite a atualização dos componentes do *Cluster* para uma nova versão sem interromper completamente os serviços. Isso é possível desde que os serviços estejam em alta disponibilidade ou que seus componentes operem em uma arquitetura distribuída em que não há um único ponto de coordenação e controle (*No Master*).

Estando todos os pré-requisitos atendidos, siga os passos a seguir:

1. Clique na botão UPGRADE da nova versão.

2. Selecione e confirme a opção "Rolling Upgrade".

3. Confirme a realização do backup dos bancos de metadados.

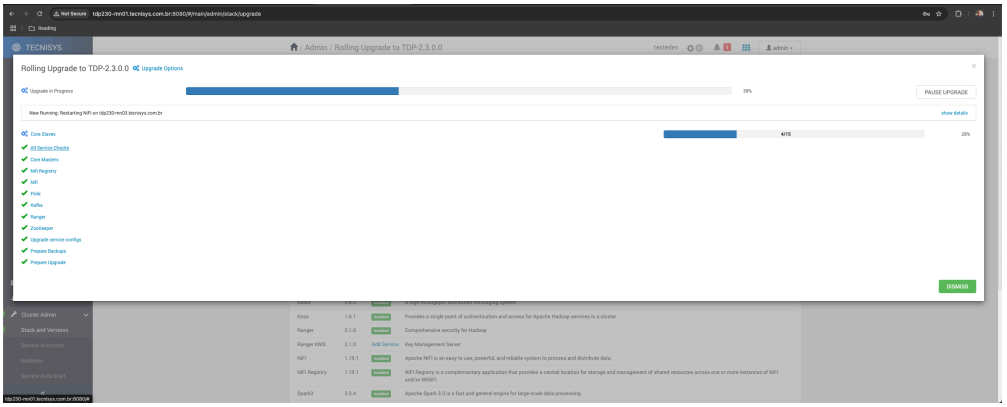


> **NOTE**

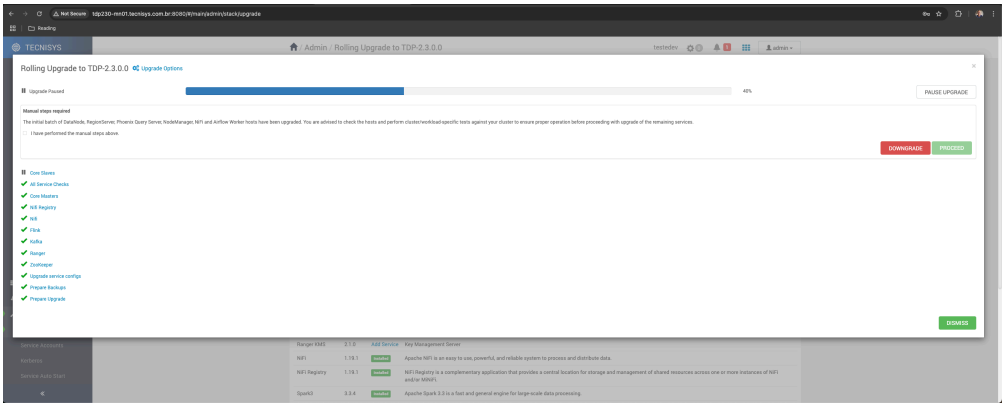
© Copyright Tecnisys 2026. All rights reserved.

Uma série de configurações serão realizadas de forma automatizada durante a atualização.

> **NOTE**
>
> A qualquer momento é possível suspender o "Rolling Upgrade". Para isto, selecione o botão "Pause Upgrade". Dessa forma, é possível pausar a atualização para corrigir qualquer irregularidade ou reverter a operação (*rollback*).



4. Confirme a realização das etapas (*steps*) manuais solicitadas.



> **TIP**
>
> É possível verificar se os componentes atualizados já estão funcionais. Para isto, clique no link "all service checks". Assim é possível identificar e corrigir eventuais problemas antes da etapa de verificação dos serviços (*Service Check*), realizada ao final da atualização.

© Copyright Tecnisys 2026. All rights reserved.

5. Acompanhe a verificação de todos os serviços atualizados (*Service Check*).

6. Selecione uma das seguintes opções para concluir o processo de atualização:

- *DOWNGRADE*: Permite reverter o processo de atualização. Todos os componentes voltam para suas versões anteriores.
- *FINALIZE LATER*: Permite finalizar o processo de atualização em outro momento, possibilitando a execução de testes e validações.
- *FINALIZE*: Finaliza o processo de atualização e define a nova versão como "CURRENT".

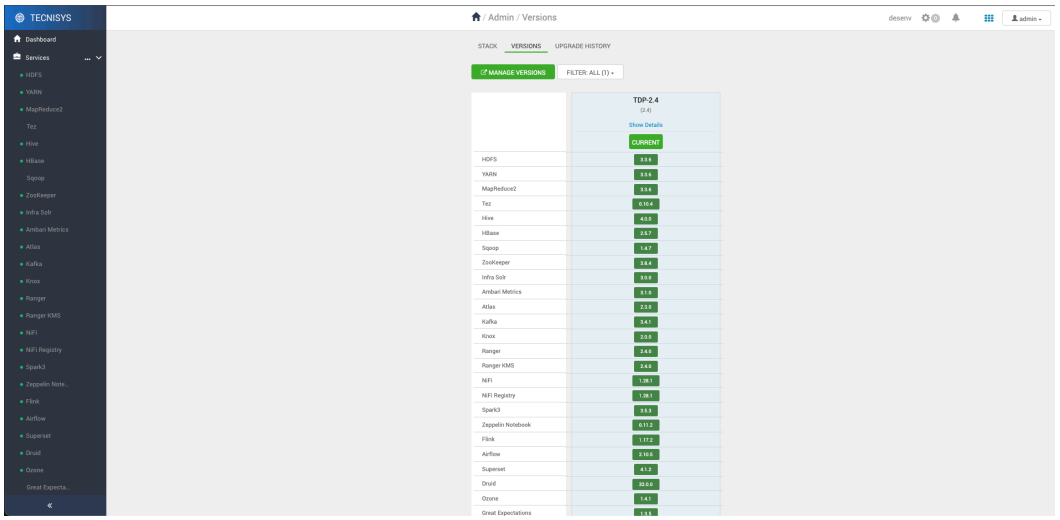
> **WARNING**
>
> Caso o Apache HBase esteja entre os serviços atualizados, ao final do *upgrade* será solicitada a confirmação da exclusão do *snapshot* criado durante o processo.

7. Selecione "Finalize" para finalizar o processo de atualização.

8. Clique em "Proceed" para confirmar a finalização do processo de atualização.

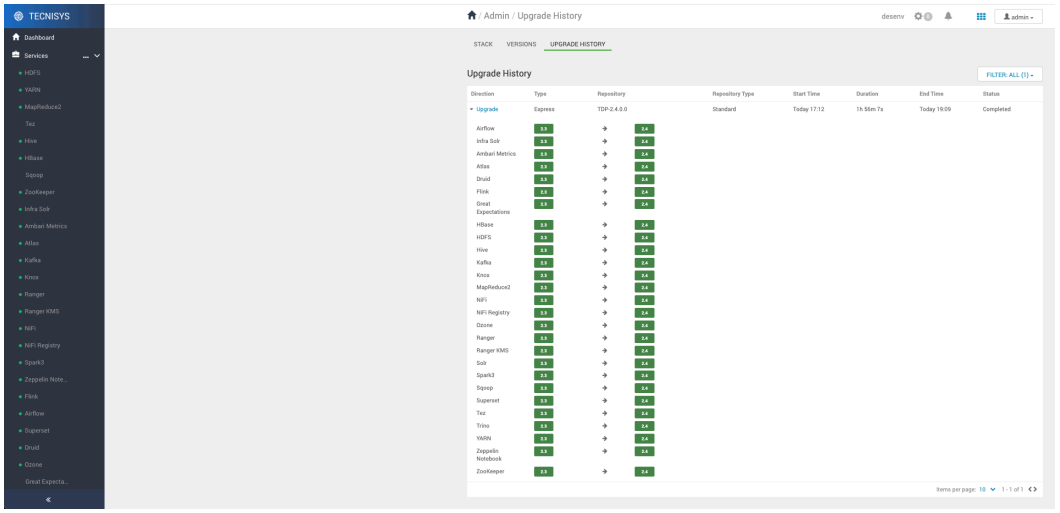
Conclusão da Atualização

Na guia VERSIONS da página "Stack and Versions" a versão anterior foi removida e a nova versão está definida como "CURRENT".

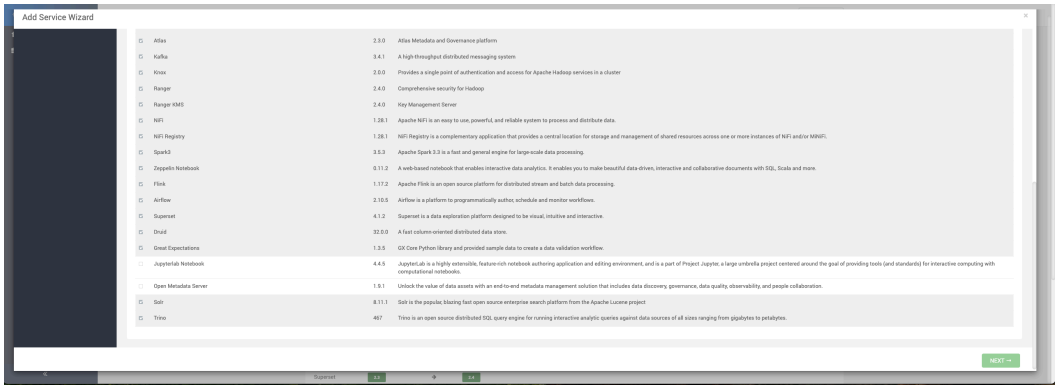


© Copyright Tecnisys 2026. All rights reserved.

Note que a guia UPGRADE HISTORY foi adicionada à página "Stack and Versions". Nesta guia é apresentado o histórico de atualizações do cluster, incluindo detalhes como o tipo de *upgrade* realizado, a versão anterior e a nova versão da *stack* e seus serviços.



Agora, na guia STACK da página "Stack and Versions" é possível adicionar os novos serviços disponíveis na nova versão do TDP.



Por fim, reinicie o serviço do Ambari Server:

Terminal input

```
ambari-server restart
```

IMPORTANT

© Copyright Tecnisys 2026. All rights reserved.



A reinicialização do Ambari Server é necessária para a consolidação de determinadas alterações e configurações.

WARNING

Após finalizar, lembre-se de reativar o Início Automático dos Serviços, selecionando, no menu lateral do Ambari, a opção "Service Auto-Start" e alterando a chave "Auto Start Settings" para "Enabled".

© Copyright Tecnisys 2026. All rights reserved.

PARTE III - TDP KUBERNETES

© Copyright Tecnisys 2026. All rights reserved.

INSTALAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Pré-Requisitos

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Esta seção descreve os pré-requisitos necessários para a instalação do **TDP Kubernetes**. Certifique-se de que todos os itens estejam atendidos antes de iniciar o processo de instalação dos componentes.

Requisitos de Infraestrutura

Orquestrador

O TDP Kubernetes é homologado nos seguintes orquestradores:

Orquestrador	Versão mínima
Kubernetes	1.32+
Red Hat OpenShift	4.19+
Rancher Manager	2.10.x+

- Cluster operacional com pelo menos 3 *worker nodes* (recomendado para alta disponibilidade)
- Acesso administrativo ao cluster via `kubectl`

Ferramentas de Linha de Comando

Ferramenta	Versão Mínima	Descrição
`kubectl`	1.27 e compatível com o cluster	CLI do Kubernetes, configurado para o cluster de destino; versão deve ser

© Copyright Tecnisys 2026. All rights reserved.

Ferramenta	Versão Mínima	Descrição
		compatível com a versão do cluster
`helm`	3.2.0+	Gerenciador de pacotes Kubernetes para instalação dos charts

Verifique as versões instaladas:

Terminal input

```
kubectl version --client
helm version
```

Classes de Armazenamento (Storage Classes)

O TDP Kubernetes requer *Storage Classes* configuradas no cluster para provisionamento dinâmico de volumes persistentes.

Uma `StorageClass` define como o cluster provisiona volumes persistentes sob demanda: ela aponta para um provisionador — como CSI de nuvem, NFS, Ceph, Longhorn ou equivalente — e determina o tipo de disco, a política de retenção e demais parâmetros operacionais.

Quando um chart do TDP solicita armazenamento via `PersistentVolumeClaim` (PVC), o Kubernetes usa a `StorageClass` indicada — ou a padrão do cluster — para criar o `PersistentVolume` correspondente no backend.

Modo de Acesso	Uso	Obrigatório
ReadWriteOnce (RWO)	Utilizado pela maioria dos componentes (PostgreSQL, Kafka, ClickHouse, etc.)	Sim

© Copyright Tecnisys 2026. All rights reserved.

Modo de Acesso	Uso	Obrigatório
`ReadWriteMany` (RWX)	Necessário quando algum componente ou modo de instalação exige volume persistente compartilhado entre múltiplos Pods	Condicional

> **NOTE**
>
> O modo `ReadWriteMany` (RWX) é necessário quando múltiplos Pods precisam acessar o mesmo volume persistente em leitura e escrita. O Apache Airflow é um exemplo comum quando DAGs ou logs são compartilhados por PVC, mas a necessidade de RWX depende da arquitetura e do modo de instalação do componente. Soluções como NFS, CephFS, Azure Files ou equivalentes podem fornecer suporte a RWX, dependendo do ambiente.

Verifique as *Storage Classes* disponíveis no cluster:

Terminal input

```
kubectl get storageclass
```

Exposição HTTP/HTTPS dos componentes

É necessário possuir uma solução de entrada de tráfego HTTP/HTTPS para expor as interfaces web de alguns componentes do TDP Kubernetes, como Airflow, Superset, Argo CD, NiFi e JupyterHub.

Ingress Controller

O ambiente pode utilizar um controlador Ingress para publicação dos serviços web no cluster.

Exemplos comuns:

© Copyright Tecnisys 2026. All rights reserved.

- NGINX Ingress Controller
- Traefik
- HAProxy
- AWS Load Balancer Controller

Verifique se há um controlador Ingress instalado no cluster. Exemplo com NGINX Ingress Controller:

Terminal input

```
kubectl get ingressclass
```

Gateway API

O ambiente também pode utilizar Gateway API, desde que o cluster possua os recursos e o controller compatível já instalados.

Exemplo de verificação:

Terminal input

```
kubectl get gatewayclass
```

> **NOTE**
>
> A API Ingress no Kubernetes está congelada e não recebe novas evoluções. Além disso, o projeto Ingress NGINX foi retirado em março de 2026. O uso de Ingress Controller ou Gateway API deve seguir a estratégia adotada para o ambiente e as implementações suportadas no cluster.

Configuração de Rede

DNS

- Os *nodes* do cluster Kubernetes devem possuir resolução DNS funcional
- Recomenda-se configurar registros DNS para os serviços expostos via *Ingress* (ex.: `airflow.tdp.exemplo.com.br`, `superset.tdp.exemplo.com.br`)

© Copyright Tecnisys 2026. All rights reserved.



Politicas de Rede (Network Policies)

- Certifique-se de que as *Network Policies* do cluster permitam a comunicação entre os *pods* do namespace.
- Os componentes do TDP precisam se comunicar livremente dentro do mesmo namespace
- Caso utilize *Network Policies* restritivas, configure regras que permitam o trafego *ingress* e *egress* entre os *pods* do TDP

  Em ambientes com políticas de conectividade restritivas, essas regras podem precisar ser aplicadas antes da instalação dos componentes ou durante o bootstrap do ambiente GitOps. Consulte Políticas de cluster para exemplos de NetworkPolicy e critérios de revisão.

  Se o ambiente também exigir autenticação para baixar imagens de um registry privado e a estratégia for centralizar `imagePullSecrets` com Kyverno, consulte Kyverno e registry privado. Essa etapa é condicional e só se aplica quando adotada pela política do cluster.

Certificados TLS/SSL (Opcional)

Para habilitar HTTPS nos serviços expostos via *Ingress*:

- Provisione certificados TLS/SSL para os domínios configurados
- Utilize o cert-manager para automação de certificados com Let's Encrypt ou CA interna
- Configure os certificados nos recursos *Ingress* de cada componente

Autenticação no Registry OCI

Para instalar o TDP Kubernetes, é necessário possuir credenciais de acesso ao registry OCI da Tecnisys, disponível em `oci://registry.tecnisys.com.br/tdp/`. Esse registry armazena e distribui os **Helm charts**[1] da plataforma.

As credenciais são usadas de formas diferentes conforme o método de instalação:

© Copyright Tecnisys 2026. All rights reserved.

Método	Como a credencial é usada
Helm CLI	O operador autentica o Helm localmente com `helm registry login`.
Argo CD	O registry é cadastrado como repositório diretamente no Argo CD.

> **NOTE**
>
> O `helm registry login` descrito abaixo é necessário apenas para instalações feitas diretamente via **Helm CLI**. Em instalações via **Argo CD**, use a mesma credencial para cadastrar o registry no Argo CD — consulte Instalação via Argo CD.

Comandos

1. **Portal Web —** Acesse o portal do registry Tecnisys com as credenciais fornecidas pela Tecnisys. Após o login, gere ou copie a credencial de uso em linha de comando, apresentada no portal como CLI Secret ou token de acesso.

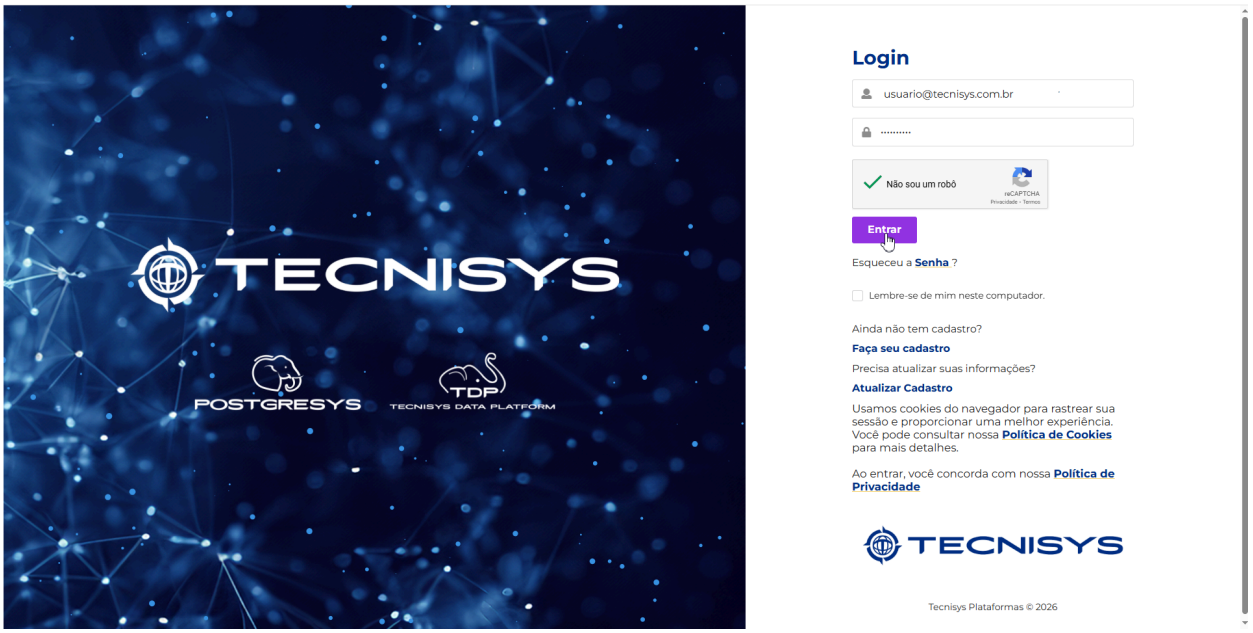


Figura 1 - Login no Registry Web

© Copyright Tecnisys 2026. All rights reserved.

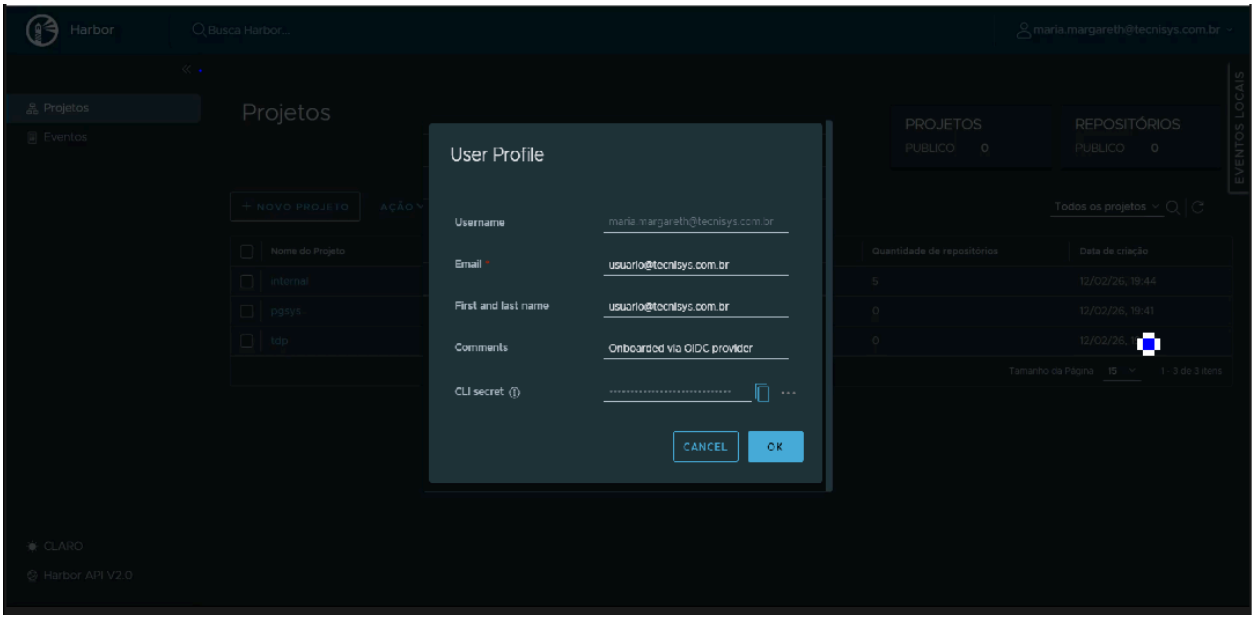


Figura 2 - Captura do CLI Secret

> **NOTE**
>
> A obtenção do CLI Secret/token é comum aos dois métodos de instalação. Em instalações via Argo CD, essa credencial será informada no cadastro do registry dentro do Argo CD. Em instalações via Helm CLI, ela será usada no `helm registry login`.

2. **Helm CLI —** Se a instalação for feita diretamente via Helm CLI, autentique o Helm no registry OCI para acessar os charts da plataforma.

Terminal input

```
helm registry login registry.tecnisys.com.br -u <usuario@tecnisys...>
```

Ao executar o comando, o terminal solicitará a senha:

```
Password:
```

Informe o **CLI Secret** / token obtido no **passo 1**.

© Copyright Tecnisys 2026. All rights reserved.

Se a autenticação for concluída com sucesso, o Helm exibirá a confirmação de login e, a partir daí, será possível consultar e instalar os charts disponíveis no registry.

> **NOTA**
>
> As credenciais de acesso ao registry devem ser obtidas junto à Tecnisys. Caso você ainda não possua acesso, entre em contato com a equipe responsável para solicitar seu usuário autorizado e a credencial de uso em CLI.

```
PS C:\> helm registry login registry.tecnisys.com.br -u usuário@tecnisys.com.br
Password:
Login Succeeded
PS C:\>
```

Figura 3 - Registry Login via Helm

3. **Validação via Helm CLI —** Após a autenticação, verifique o acesso listando um chart de exemplo:

Terminal input

```
helm show chart oci://registry.tecnisys.com.br/tdp/charts/tdp-spark
```

```
PS C:\lab-gitops> helm show chart oci://registry.tecnisys.com.br/tdp/charts/tdp-spark
Pulled: registry.tecnisys.com.br/tdp/charts/tdp-spark:3.0.0
Digest: sha256:4d3eb93af86806760f45e84d0dc4f6aa5038dcfbfbb44b52fe8e71b3310a1193
apiVersion: v2
appVersion: 4.0.0
dependencies:
- condition: spark.enabled
  name: spark
  repository: oci://registry.tecnisys.com.br/upstream/spark
  version: '*'
description: TDP Spark chart for distributed data processing on Kubernetes
home: https://github.com/user/tdp-k8s
keywords:
- spark
- analytics
- big data
- distributed computing
maintainers:
- email: suporte.tdp@tecnisys.com.br
  name: Tecnisys
name: tdp-spark
type: application
version: 3.0.0

PS C:\lab-gitops>
```

Figura 4 - Show chart - exemplo

Lista de Verificação

Antes de prosseguir para a instalação, confirme todos os itens abaixo:

© Copyright Tecnisys 2026. All rights reserved.

- [ ] Cluster Kubernetes 1.32+ (ou Red Hat OpenShift 4.19+, ou Rancher Manager 2.10.x+) operacional
- [ ] `kubectl` configurado e conectado ao cluster
- [ ] `helm` 3.2.0+ instalado
- [ ] *Storage Class* com suporte a **ReadWriteOnce** disponível
- [ ] *Storage Class* com suporte a `ReadWriteMany` (RWX), quando algum componente ou modo de instalação exigir volume persistente compartilhado entre múltiplos Pods
- [ ] Solução de entrada HTTP/HTTPS (Ingress Controller ou Gateway API) instalada e operacional
- [ ] Recursos computacionais suficientes nos *nodes* do cluster
- [ ] Resolução DNS configurada para os *nodes*
- [ ] Políticas de cluster e/ou Kyverno avaliados e aplicados, quando exigidos pelo ambiente
- [ ] Credenciais de acesso ao registry OCI da Tecnisys obtidas
- [ ] Autenticação via `helm registry login registry.tecnisys.com.br` realizada com sucesso, se a instalação for via Helm CLI
- [ ] Certificados TLS/SSL provisionados (opcional)

Footnotes

1. **Helm chart**: pacote que reúne templates de manifestos Kubernetes e valores por omissão; o Helm aplica esse pacote no cluster para criar ou atualizar os recursos do componente, ajustáveis via arquivos de valores ou flags. ↩

© Copyright Tecnisys 2026. All rights reserved.

Instalação via Helm

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Esta seção descreve o processo de instalação dos componentes do TDP Kubernetes utilizando o Helm CLI.

Nessa abordagem, os componentes são instalados por meio da execução direta de comandos helm install, que implantam no cluster Kubernetes os recursos definidos em cada Helm Chart disponível no registry OCI privado da Tecnisys.

Na instalação via Helm a aplicação dos charts é realizada manualmente ou por automação externa, comando a comando, com controle direto sobre a sequência de implantação de cada componente.

> **TIP**
>
> Antes de iniciar a instalação, certifique-se de que todos os pré-requisitos foram atendidos, incluindo a autenticação no registry OCI da Tecnisys.

Ordem de Instalação

A instalação dos componentes deve seguir uma ordem específica para garantir que as dependências sejam atendidas. A tabela abaixo apresenta a ordem recomendada:

Ordem	Chart	Componente	Versão	Dependências
1	`tdp-crds`	CRDs da Plataforma	3.0.1	Nenhuma
2	`tdp-argo`	Argo CD	3.2.5	tdp-crds

© Copyright Tecnisys 2026. All rights reserved.

Ordem	Chart	Componente	Versão	Dependências
3	`tdp-postgresql`	PostgreSQL	17.5.0	Nenhuma
4	`tdp-kafka`	Kafka (Strimzi)	4.1.0	tdp-crds (Strimzi CRDs)
5	`tdp-hive-metastore`	Hive Metastore	4.0.0	tdp-postgresql
6	`tdp-deltalake`	Delta Lake	4.0.1	Nenhuma
7	`tdp-iceberg`	Iceberg	1.10.0	Nenhuma
8	`tdp-ozone`	Apache Ozone	2.0.0	Nenhuma
9	`tdp-ranger`	Ranger	2.7.0	tdp-postgresql
10	`tdp-nifi`	NiFi	1.28.0	Nenhuma
11	`tdp-spark`	Spark	4.0.2	Nenhuma
12	`tdp-trino`	Trino	478	Nenhuma
13	`tdp-airflow`	Airflow	3.0.2	tdp-postgresql
14	`tdp-jupyter`	JupyterHub	5.3.0	tdp-postgresql

© Copyright Tecnisys 2026. All rights reserved.



Ordem	Chart	Componente	Versão	Dependências
15	`tdp-clickhouse`	ClickHouse	25.8.11.66	tdp-crds (Altinity CRDs)
16	`tdp-cloudbeaver`	CloudBeaver	25.3.0	tdp-postgresql
17	`tdp-openmetadata`	OpenMetadata	1.12.4	tdp-postgresql
18	`tdp-superset`	Superset	5.0.0	tdp-postgresql

A coluna **Versão** refere-se à versão do **componente** (`appVersion` do chart, ex.: Kafka 4.1.0, Argo CD 3.2.5). A versão do **chart** Helm (`version` no `Chart.yaml`) é um número separado — para o `tdp-kafka`, por exemplo, a chart version é 3.0.1 enquanto a appVersion é 4.1.0. Ao omitir `--version` no `helm install`, o Helm instala a versão mais recente disponível no registry.



WARNING

- O `tdp-postgresql` **é uma dependência** para componentes configurados para usar o PostgreSQL da stack e deve estar operacional **antes** deles.
- **Instalar um componente dependente sem o PostgreSQL ativo resultará em falha**.
- Conforme recomendado para **ambientes de produção**, componentes que dependem de banco de dados devem utilizar uma **base externa**.
- No entanto, **se nenhuma base externa for informada**, o componente utilizará uma **instância interna exclusiva** quando essa opção existir no chart.

© Copyright Tecnisys 2026. All rights reserved.

- Para entender melhor essa diferença, consulte PostgreSQL interno versus externo.

Instalação dos Componentes

Comandos

Instalação

O padrão de instalação é o mesmo para todos os componentes. Para cada chart, execute:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  --create-namespace \
  -f <VALUES_FILE>
```

```
PS C:\lab-gitops> helm install demo-postgresql oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql -n tdp-install-doc --create-namespace
Pulled: registry.tecnisys.com.br/tdp/charts/tdp-postgresql:3.0.0
Digest: sha256:accc1eab0a3199e335f43b335fd8d8d79765c1f08ecafd7920b1a803b4fa0a6e
NAME: demo-postgresql
LAST DEPLOYED: Tue Mar 31 11:49:29 2026
NAMESPACE: tdp-install-doc
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
PostgreSQL has been deployed successfully.

## Access from inside the cluster

The PostgreSQL service can be accessed on port 5432 using:
  demo-postgresql.tdp-install-doc.svc.cluster.local

## Get the postgres password

export POSTGRES_PASSWORD=$(kubectl get secret --namespace tdp-install-doc demo-postgresql -o jsonpath="{.data['postgres-password']}" | base64 -d)

## Connect and test (psql)

kubectl run demo-postgresql-client --rm --tty -i --restart='Never' --namespace tdp-install-doc \
  --image registry.tecnisys.com.br/community/images/bitnamilegacy/postgresql:17.5.0-debian-12-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
  --command -- psql -h demo-postgresql -p 5432 -U postgres -d postgres

## Access from outside the cluster (port-forward)

kubectl port-forward --namespace tdp-install-doc svc/demo-postgresql 5432:5432 &
psql -h 127.0.0.1 -p 5432 -U postgres -d postgres
PS C:\lab-gitops>
```

Figura 1 - Instalação via Helm

NOTE

© Copyright Tecnisys 2026. All rights reserved.

O parâmetro `--create-namespace` cria o namespace `<NAMESPACE>` automaticamente na primeira instalação. Nas instalações subsequentes, o namespace já existirá e o parâmetro será ignorado.

Se a flag `--version` for omitida, o Helm instalará a versão mais recente do chart disponível no registry. Para fixar uma versão específica do chart, utilize `--version <CHART_VERSION>`.
A versão do componente instalado (Kafka, Airflow, etc.) é definida pelo `appVersion` do chart e pode ser consultada no próprio chart antes da instalação ou verificada nos recursos implantados após a instalação.

VERIFICAÇÃO APÓS CADA INSTALAÇÃO

Após instalar cada componente, aguarde os pods ficarem prontos antes de prosseguir:

Terminal input

```
kubectl get pods -n <NAMESPACE> --watch
```



Figura 2 - Verifica os Pods

Pressione `Ctrl+C` quando todos os pods do componente estiverem com status `Running` e `READY`.

1. CRDs da Plataforma (`tdp-crds`)

O QUE SÃO CRDS?

Custom Resource Definitions (CRDs) ensinam ao Kubernetes novos tipos de recurso que ele não conhece nativamente. Os operadores da plataforma (NiFiKop, Strimzi, ArgoCD, ClickHouse etc.) precisam que esses tipos estejam registrados no cluster **antes** de serem instalados — sem eles, o Kubernetes não consegue criar objetos como `NifiCluster`, `Kafka` ou `Application`.

© Copyright Tecnisys 2026. All rights reserved.



Os CRDs são recursos de escopo global (não pertencem a um namespace) e são gerenciados num chart dedicado (`tdp-crds`) para evitar conflitos durante upgrades do Helm.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  --create-namespace \
  -f <VALUES_FILE>
```

NOME DO RELEASE (`<RELEASE_NAME>`)

Nos comandos de instalação abaixo, `<RELEASE_NAME>` é o nome do release Helm do componente. Muitas equipes usam o prefixo `tdp-` alinhado ao nome do chart (ex.: chart `tdp-kafka` com release `tdp-kafka`). Ajuste à convenção do seu ambiente.

CRDS INCLUÍDOS NO `tdp-crds`

Este chart instala os CRDs necessários para:

- **Argo CD** — `Application`, `ApplicationSet`, `AppProject`
- **Strimzi Kafka Operator** — `Kafka`, `KafkaConnect`, `KafkaTopic`, `KafkaUser`, `KafkaNodePool`, entre outros
- **NiFiKop Operator** — `NifiCluster`, `NifiDataflow`, `NifiUser`, entre outros
- **Altinity ClickHouse Operator** — `ClickHouseInstallation`, `ClickHouseKeeperInstallation`, entre outros
- **OpenMetadata OMJob Operator** — `OMJob`, `CronOMJob`
- **cert-manager** — `Certificate`, `ClusterIssuer`, `Issuer`, entre outros

2. ArgoCD

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  --create-namespace \
  -f <VALUES_FILE>
```

3. PostgreSQL

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

> **IMPORTANT**
>
> Aguarde o PostgreSQL ficar totalmente operacional antes de prosseguir com a instalação dos componentes que dependem dele. Verifique o status:
>
> Terminal input
>
> ```
> kubectl get pods -n <NAMESPACE> | grep <RELEASE_NAME>
> ```

O *pod* deve apresentar o status `Running` e estar com todas as réplicas prontas (`READY 1/1`).

> **EXEMPLO**
>
> Se o release do PostgreSQL for `tdp-postgresql`, o filtro pode ser `grep tdp-postgresql` na saída de `kubectl get pods`.

4. Kafka (Strimzi)

© Copyright Tecnisys 2026. All rights reserved.

PRÉ-REQUISITO: CRDS DO STRIMZI

O `tdp-kafka` depende dos **CRDs do Strimzi** (ex.: `Kafka`, `KafkaNodePool`, `KafkaConnect`, `KafkaConnector`) que são instalados pelo `tdp-crds` (passo 1). Certifique-se de que o `tdp-crds` foi aplicado **antes** de instalar o `tdp-kafka`. Sem os CRDs, o Helm não conseguirá criar os recursos do Strimzi e a instalação falhará.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  --create-namespace \
  -f <VALUES_FILE>
```

VALIDAÇÃO PÓS-INSTALAÇÃO DO KAFKA

Após a instalação, verifique se o cluster Kafka foi criado corretamente:

Terminal input

```
kubectl get kafka -n <NAMESPACE>
kubectl get kafkanodepools -n <NAMESPACE>
kubectl get pods -n <NAMESPACE> -l strimzi.io/cluster=
<KAFKA_CLUSTER_NAME>
kubectl get svc -n <NAMESPACE> | grep kafka
kubectl get kafkaconnect -n <NAMESPACE>
kubectl get kafkaconnector -n <NAMESPACE>
```

Substitua `<KAFKA_CLUSTER_NAME>` pelo valor de `name` no `values.yaml` (padrão: `tdp-kafka`).

CHART VERSION VS. APPVERSION

© Copyright Tecnisys 2026. All rights reserved.



> Para o `tdp-kafka`, a **chart version** (ex.: `3.0.1`) é diferente da **appVersion** (ex.: `4.1.0` — versão do Apache Kafka).
> Use `helm show chart oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka` para consultar ambas.

5. Hive Metastore

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

6. Delta Lake

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

7. Iceberg

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

8. Apache Ozone

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

9. Ranger

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

10. NiFi

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-nifi \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

11. Spark

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-spark \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

12. Trino

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-trino \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

13. Airflow

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

> **NOTE**
>
> O Airflow requer que o PostgreSQL esteja operacional. Além disso, caso utilize o executor `CeleryExecutor` ou `KubernetesExecutor` com compartilhamento de DAGs via volume, certifique-se de que uma *Storage Class* com suporte a **ReadWriteMany** (RWX) esteja disponível.

14. JupyterHub

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

15. ClickHouse

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

16. CloudBeaver

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

17. OpenMetadata

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE> \
  --wait --wait-for-jobs --timeout 10m
```

OPENMETADATA — JOBS DE INICIALIZAÇÃO

O OpenMetadata executa Jobs Kubernetes durante a instalação e atualização. O flag `--wait-for-jobs` instrui o Helm a aguardar a conclusão desses jobs antes de considerar a instalação bem-sucedida.
Recomenda-se usar `--wait --wait-for-jobs --timeout 10m` para garantir que os jobs pós-instalação sejam concluídos.

© Copyright Tecnisys 2026. All rights reserved.



> No Argo CD, a Application só deve ser considerada saudável após os jobs pós-instalação/atualização serem concluídos.

18. Superset

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-superset \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

Verificação da Instalação

Comandos

Verificação da Instalação

Após instalar todos os componentes desejados, verifique o status geral da instalação.

Verificar os Pods

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

```
PS C:\lab-gitops> kubectl get pods -n tdp-install-doc
NAME                              READY   STATUS    RESTARTS   AGE
demo-postgresql-tdp-postgresql-0   1/1     Running   0          3m57s
PS C:\lab-gitops>
```

Figura 1 - Get Pods

Todos os *pods* devem apresentar o status `Running` e estar com todas as replicas prontas.

Verificar os Releases do Helm

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm list -n <NAMESPACE>
```

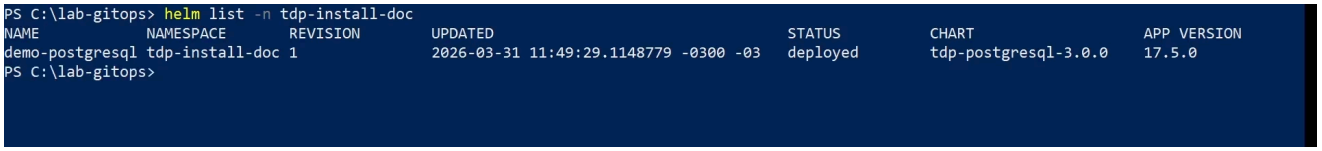


Figura 2 - Helm Releases

Todos os releases devem apresentar o status `deployed`.

Verificar os Serviços

Terminal input

```
kubectl get svc -n <NAMESPACE>
```

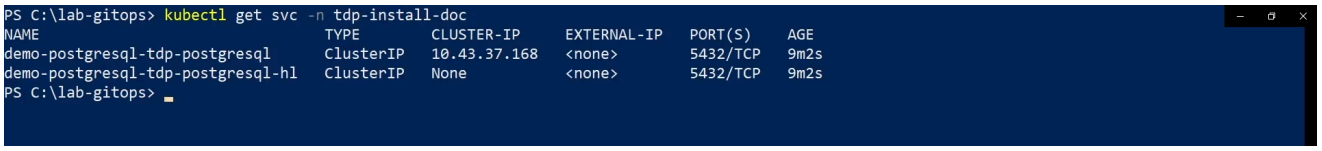


Figura 3 - Get svc

Verificar os Volumes Persistentes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```



Figura 4 - Get pvc

Todos os *Persistent Volume Claims* devem apresentar o status `Bound`.

Resolução de Problemas

Pods Não Iniciam

© Copyright Tecnisys 2026. All rights reserved.

Se um *pod* permanece no status `Pending` ou `CrashLoopBackOff`:

1. Verifique os eventos do *pod*:

Terminal input

```
kubectl describe pod <pod-name> -n <NAMESPACE>
```

2. Verifique os logs do *pod*:

Terminal input

```
kubectl logs <pod-name> -n <NAMESPACE>
```

3. Causas comuns:

- **Recursos insuficientes:** o cluster não possui CPU ou memória suficiente para agendar o *pod*. Verifique com `kubectl describe node`
- **PVC pendente:** a *Storage Class* configurada não está disponível ou não suporta o modo de acesso solicitado
- **Dependência não atendida:** o componente depende do PostgreSQL ou de outro serviço que ainda não está operacional

Erro de Pull de Imagem (ImagePullBackOff)

Se um *pod* apresenta o status `ImagePullBackOff` ou `ErrImagePull`:

1. Verifique se a autenticação no registry foi realizada corretamente:

Terminal input

```
helm registry login registry.tecnisys.com.br
```

2. Verifique se o *Secret* de autenticação do registry está configurado no namespace:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl get secrets -n <NAMESPACE> | grep registry
```

3. Caso necessário, crie o *Secret* manualmente:

Terminal input

```
kubectl delete secret <docker-registry-secret> -n <NAMESPACE> 2>/dev/null ||
true
kubectl create secret docker-registry <docker-registry-secret> \
  --docker-server=registry.tecnisys.com.br \
  --docker-username=<username> \
  --docker-password=<password> \
  -n <NAMESPACE>
```

EXEMPLO

Exemplo

```
kubectl delete secret tdp-registry-secret -n <NAMESPACE> 2>/dev/null ||
true
kubectl create secret docker-registry tdp-registry-secret \
  --docker-server=registry.tecnisys.com.br \
  --docker-username=<username> \
  --docker-password=<password> \
  -n <NAMESPACE>
```

Falha na Conexão com o Banco de Dados

Se componentes como Airflow, Superset, Ranger, OpenMetadata ou CloudBeaver apresentam erros de conexão com o banco de dados:

1. Verifique se o PostgreSQL está operacional:

Terminal input

```
kubectl get pods -n <NAMESPACE> | grep <RELEASE_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.



> **EXEMPLO**
>
> Com release `tdp-postgresql`: `kubectl get pods -n <NAMESPACE> | grep tdp-postgresql`

2. Verifique a conectividade interna do cluster:

Terminal input

```
kubectl run pg-test --rm -it --image=busybox -n <NAMESPACE> -- \
  sh -c "nc -zv <postgresql-service> 5432"
```

> **EXEMPLO**
>
> Quando o serviço padrão do chart expõe o host `tdp-postgresql` no namespace:
>
> **Exemplo**
>
> ```
> kubectl run pg-test --rm -it --image=busybox -n <NAMESPACE> -- \
>   sh -c "nc -zv tdp-postgresql 5432"
> ```

3. Verifique as credenciais configuradas nos *values* do chart do componente

Desinstalar um Componente com Problema

Para desinstalar e reinstalar um componente especifico:

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

© Copyright Tecnisys 2026. All rights reserved.



WARNING

A desinstalação de um release Helm **não remove** os *Persistent Volume Claims* (PVCs) associados. Para uma reinstalação limpa, exclua os PVCs manualmente antes de reinstalar:

Terminal input

```
kubectl delete pvc <pvc-name> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Argo CD

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

> **TIP**
>
> Antes de começar, confirme que os pré-requisitos foram atendidos.
>
> Se o Argo CD ainda **não estiver instalado**, o **Passo 4** abaixo (Opção A, `deploy.sh --install`) o instala automaticamente — junto com o bootstrap do GitOps. Se **já estiver instalado**, use a **Opção B** (`deploy.sh -c`).
>
> Para instalar os componentes **manualmente**, sem GitOps, consulte a instalação via Helm.

Visão geral

O **Argo CD** instala componentes a partir de arquivos YAML chamados **Applications**. Cada componente do TDP — PostgreSQL, Kafka, Airflow, Trino, etc. — é representado por um arquivo Application que instrui o Argo CD a baixar o chart correspondente do registry da Tecnisys e instalá-lo no cluster.

Esses arquivos ficam em um repositório Git. O Argo CD monitora esse repositório continuamente e sincroniza o estado do cluster com o que está declarado — sem precisar rodar comandos a cada mudança.

Diferença entre instalação com Argo CD e Helm

Critério	Helm CLI	Argo CD (GitOps)
Rastreabilidade	Comandos manuais, sem histórico automático	Cada mudança fica registrada no Git

© Copyright Tecnisys 2026. All rights reserved.

Critério	Helm CLI	Argo CD (GitOps)
Automação	Execução manual de cada chart	Sincronização automática contínua
Rollback	Requer reexecução de comandos	Um clique ou um comando
Visibilidade	Somente via terminal	Interface web com status em tempo real
Auditoria	Difícil	Completa, via histórico do Git

Estrutura do repositório GitOps

A implementação atual utiliza uma estrutura de diretórios otimizada para GitOps:

```
tdp-gitops/
├── common/                          # Recursos de bootstrap (AppProject,
Secrets, App of Apps)
│   ├── argo-gitops-appproject.yaml
│   ├── argo-gitops-app-of-apps.yaml
│   ├── tdp-devops-repo-secret.yaml
│   └── tdp-registry-secret.yaml
├── available/                       # Templates das Applications (com
variáveis)
│   ├── tdp-postgresql/
│   │   ├── tdp-postgresql.yaml
│   │   └── values.yaml
│   ├── tdp-kafka/
│   └── ...
├── current/                         # Manifests renderizados (commitados no
git)
│   ├── common/
│   ├── tdp-postgresql/
│   └── ...
├── variables.env                    # Variáveis de ambiente (não commitar)
```

© Copyright Tecnisys 2026. All rights reserved.

```
├── deploy.sh                          # Script de automação
└── README.md
```

Como funciona:

- `available/` contém **templates** com variáveis (`${VAR}`) — nunca aplicados diretamente.
- `deploy.sh` renderiza `available/` + `common/` → `current/` substituindo as variáveis.
- `current/` é **commitado no git** e monitorado pelo App of Apps do Argo CD.
- `common/` contém os recursos de bootstrap (AppProject, Secrets, App of Apps).

Conceitos essenciais

Manifest: Um arquivo YAML que descreve um recurso Kubernetes ou Argo CD. Neste contexto, cada manifest define uma **Application** do Argo CD — ou seja, instrui o Argo CD a instalar um determinado chart Helm.

Application: O objeto central do Argo CD. Orienta o Argo CD a baixar um determinado chart Helm do Registry e instalá-lo em um determinado namespace do cluster. Cada componente do TDP é representado por uma Application separada.

Repository (no contexto do Argo CD): A origem dos artefatos que o Argo CD irá utilizar. Pode ser um repositório Git (onde ficam os manifests) ou um registry OCI[1] (onde ficam os Helm Charts da Tecnisys).

Sincronização: O processo pelo qual o Argo CD compara o estado declarado nos manifests com o estado real do cluster e aplica as mudanças necessárias.

App of Apps: Um padrão que permite gerenciar todos os componentes com uma única Application raiz, que descobre e instala automaticamente todas as demais.

Ordem de instalação

Quando o App of Apps é aplicado, o Argo CD tenta sincronizar todos os componentes ao mesmo tempo.
Por isso, é importante que os componentes sem dependências estejam estáveis antes que os dependentes tentem se conectar a eles.
A tabela abaixo indica essa ordem e as dependências de cada chart:

© Copyright Tecnisys 2026. All rights reserved.

Ordem	Chart	Componente	Versão	Dependências
1	`tdp-postgresql`	PostgreSQL	17.5.0	Nenhuma
2	`tdp-kafka`	Kafka (Strimzi)	4.1.0	tdp-crds (Strimzi CRDs)
3	`tdp-hive-metastore`	Hive Metastore	4.0.0	tdp-postgresql
4	`tdp-deltalake`	Delta Lake	4.0.1	Nenhuma
5	`tdp-iceberg`	Iceberg	1.10.0	Nenhuma
6	`tdp-ozone`	Apache Ozone	2.0.0	Nenhuma
7	`tdp-ranger`	Ranger	2.7.0	tdp-postgresql
8	`tdp-nifi`	NiFi	1.28.0	Nenhuma
9	`tdp-spark`	Spark	4.0.2	Nenhuma
10	`tdp-trino`	Trino	478	Nenhuma
11	`tdp-airflow`	Airflow	3.0.2	tdp-postgresql
12	`tdp-jupyter`	JupyterHub	5.3.0	tdp-postgresql

© Copyright Tecnisys 2026. All rights reserved.

Ordem	Chart	Componente	Versão	Dependências
13	`tdp-clickhouse`	ClickHouse	25.8.11.66	tdp-crds (Altinity CRDs)
14	`tdp-cloudbeaver`	CloudBeaver	25.3.0	tdp-postgresql
15	`tdp-openmetadata`	OpenMetadata	1.12.4	tdp-postgresql
16	`tdp-superset`	Superset	5.0.0	tdp-postgresql

A coluna **Versão** refere-se à versão do **componente** (ex.: Kafka 4.1.0, Airflow 3.0.2). A versão do **chart** Helm é 3.0.1 para todos os charts TDP.

Pré-requisitos

Ferramenta	Obrigatório para
`kubectl` (configurado para o cluster)	deploy, bootstrap
`envsubst` (pacote `gettext`)	renderização de templates
`git`	modo GitOps (push de `current/`)
`helm`	bootstrap do Argo CD (`--install`)

Passos para Instalação

Comandos

Passo 1 – Obter o repositório GitOps

© Copyright Tecnisys 2026. All rights reserved.

Clone (ou faça fork) do repositório GitOps e entre na pasta:

Terminal input

```
git clone <GITOPS_REPO_URL>
cd tdp-gitops
```

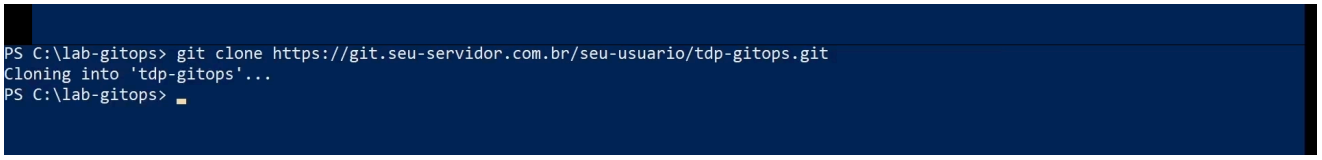


Figura 1 - Clone do repositório GitOps

Passo 2 – Configurar variáveis de ambiente

Copie o arquivo de referência e edite com seus valores:

Terminal input

```
cp variables.env variables.env.local
vim variables.env.local
```

Variáveis obrigatórias em `variables.env.local`:

variables.env.local

```
# Namespace onde o Argo CD está instalado
ARGOCD_NAMESPACE=argocd

# Namespace alvo dos pods TDP
TDP_NAMESPACE=<NAMESPACE>

# Namespace onde as Applications Argo CD são criadas
TDP_PROJECT_NAMESPACE=<NAMESPACE>

# Nome da Application "App of Apps"
TDP_APPLICATIONS=argo-gitops-tdp-project

# URL deste repositório git (usado pelo Argo CD para monitorar current/)
GIT_REPO_URL=<GITOPS_REPO_URL>
```

© Copyright Tecnisys 2026. All rights reserved.

```
GIT_REPO_USER=<GIT_USERNAME>
GIT_REPO_TOKEN_OR_PASS=<GIT_TOKEN>

# Helm OCI registry com os charts TDP
HELM_CHART_REPO_URL=registry.tecnisys.com.br/tdp/charts
HELM_CHART_VERSION=<TDP_K8S_RELEASE>

# Servidor Kubernetes (usar padrão para in-cluster)
KUBERNETES_SERVER=https://kubernetes.default.svc

# Credenciais do Helm OCI registry
TECNISYS_HELM_REGISTRY_USER=<REGISTRY_USERNAME>
TECNISYS_HELM_REGISTRY_TOKEN=<REGISTRY_CLI_SECRET>
```

Use em `HELM_CHART_VERSION` a versão da release atual: 3.0.1.

> **WARNING**
>
> `variables.env.local` está no `.gitignore` — **nunca** commite tokens ou senhas.

Passo 3 – Configurar o Argo CD para gerenciar Applications em namespace separado

As Applications ficam no namespace `<NAMESPACE>` (definido em `TDP_PROJECT_NAMESPACE`), **não** no namespace `argocd`. Configure o Argo CD para gerenciar esse namespace por meio do parâmetro `application.namespaces` — consulte Configuração do Argo CD → Application namespaces.

Passo 4 – Bootstrap do Argo CD

O script `deploy.sh` automatiza a instalação do Argo CD e o bootstrap do GitOps.

Opção A: Bootstrap completo — instala o Argo CD (recomendado)

Instala o Argo CD (`tdp-crds` + `tdp-argo`) via Helm OCI, aguarda os pods prontos, renderiza os templates e aplica os recursos comuns:

Terminal input

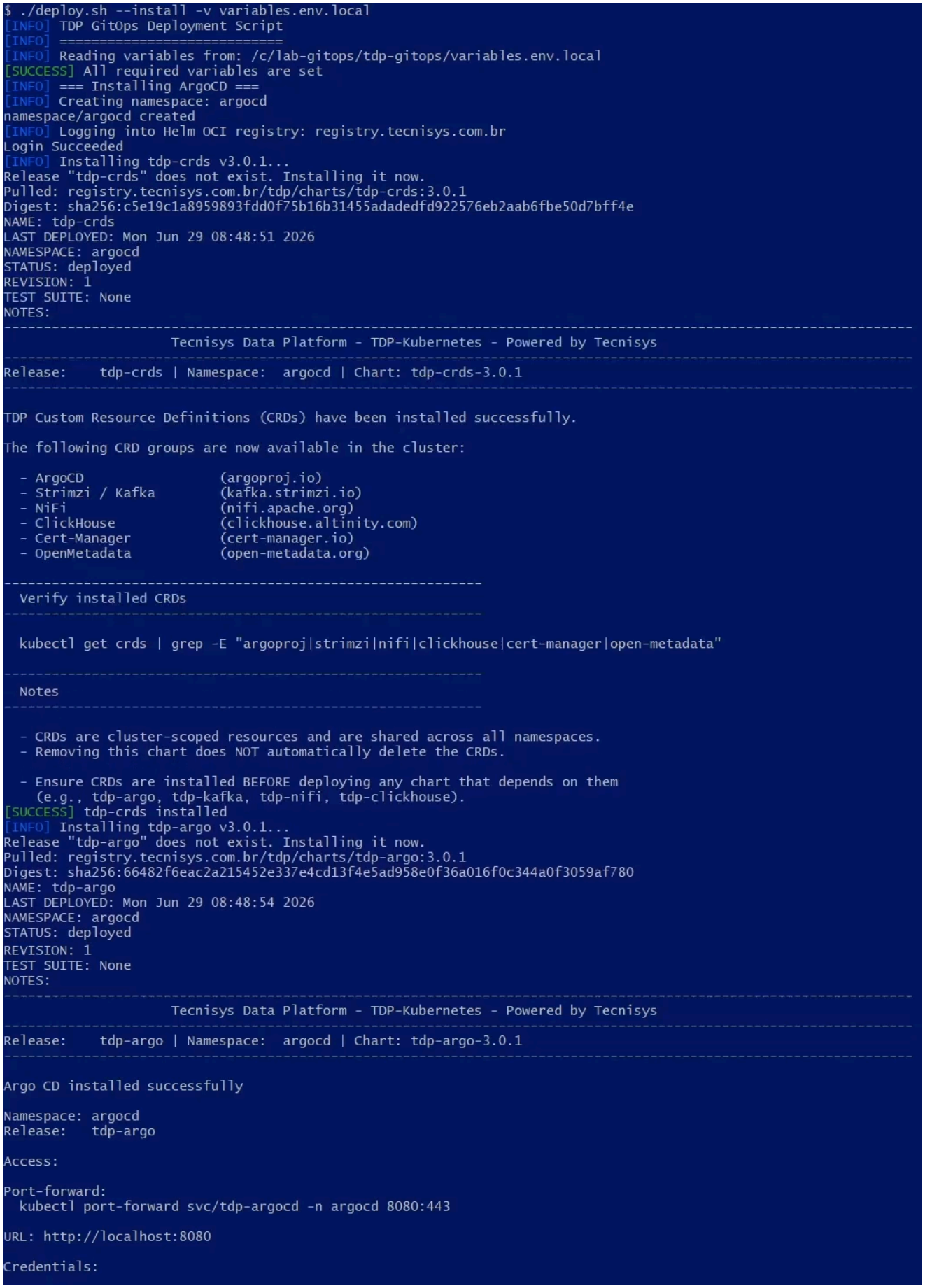
© Copyright Tecnisys 2026. All rights reserved.

```
./deploy.sh --install -v variables.env.local
```

O que é executado:

1. `helm registry login` → autentica no OCI registry
2. `helm upgrade --install tdp-crds` → instala os CRDs do Argo CD
3. `helm upgrade --install tdp-argo` → instala o Argo CD
4. `kubectl rollout status` → aguarda `argocd-server` e `application-controller`
5. `envsubst` em todos os templates → popula `current/`
6. `kubectl apply` nos arquivos de `current/common/`:
   - `argo-gitops-appproject.yaml` — AppProject TDP
   - `tdp-devops-repo-secret.yaml` — credencial git para o Argo CD
   - `tdp-registry-secret.yaml` — credencial do Helm OCI registry
   - `argo-gitops-app-of-apps.yaml` — Application "App of Apps"

© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.

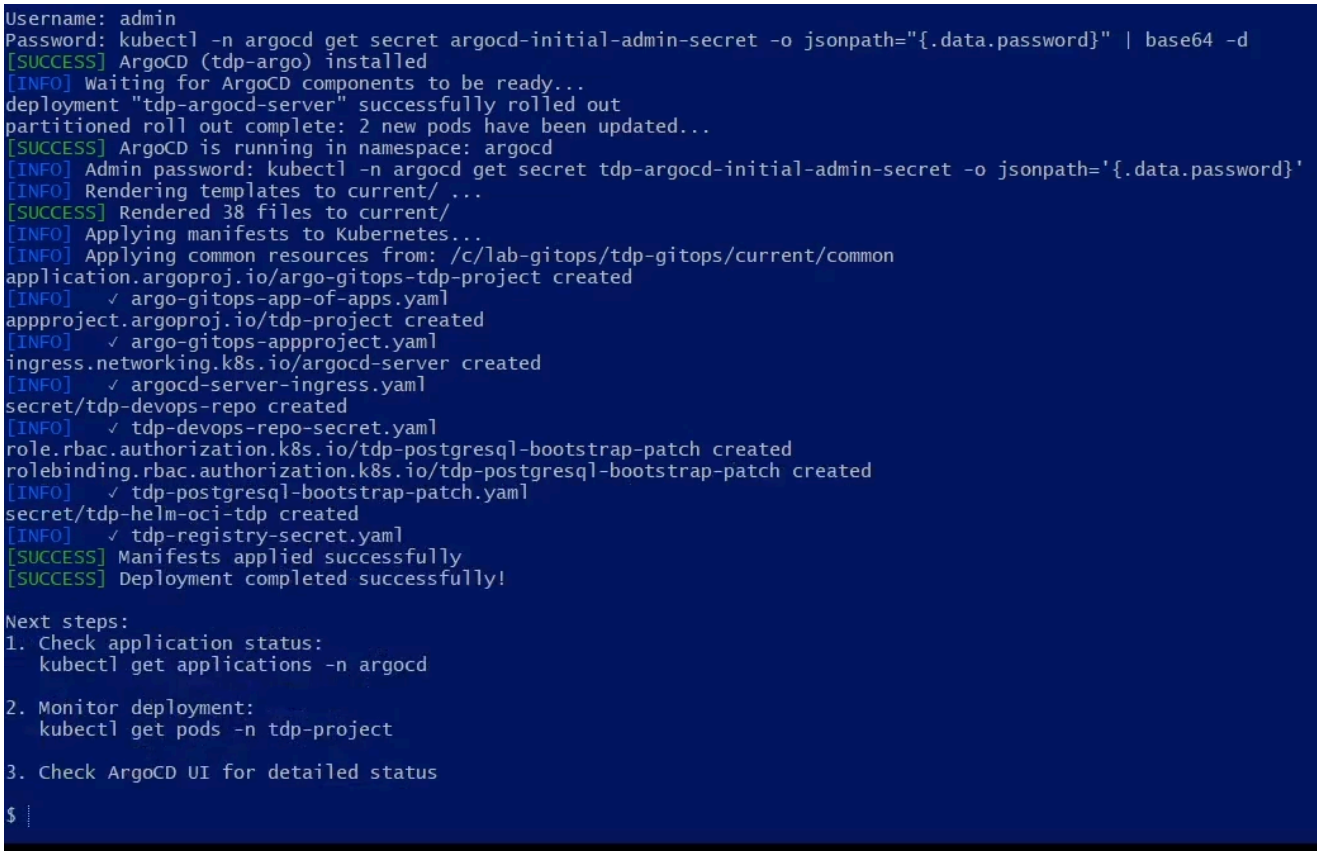


Figura 2 - Bootstrap completo com deploy.sh --install

Opção B: Apenas o GitOps — Argo CD já instalado

Se o Argo CD já estiver instalado, apenas renderize e aplique os recursos comuns:

Terminal input

```
./deploy.sh -c -v variables.env.local
```

© Copyright Tecnisys 2026. All rights reserved.

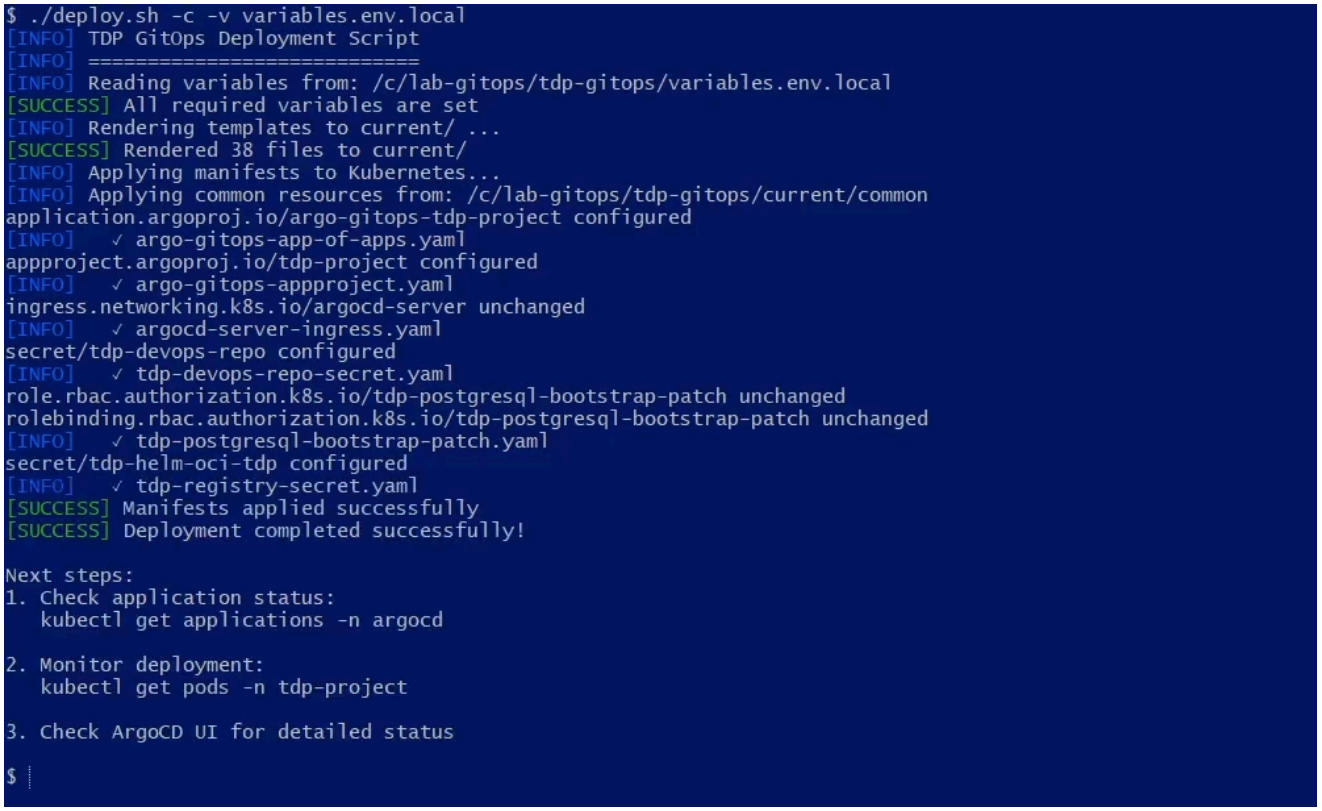


Figura 3 - Bootstrap manual com deploy.sh -c

Passo 5 – Commitar e publicar `current/` no Git

O App of Apps lê `current/` **diretamente do repositório git**. Os arquivos precisam estar publicados:

Terminal input

```
git add current/
git commit -m "chore: render gitops manifests"
git push origin main
```

O App of Apps criará automaticamente todas as Applications presentes em `current/` com `selfHeal: true`.

© Copyright Tecnisys 2026. All rights reserved.

```
$ git add current/
git commit -m "chore: render gitops manifests"
git push -u origin main
[main (root-commit) 25d4e43] chore: render gitops manifests
 7 files changed, 492 insertions(+)
 create mode 100644 current/.gitkeep
 create mode 100644 current/common/argo-gitops-app-of-apps.yaml
 create mode 100644 current/common/argo-gitops-appproject.yaml
 create mode 100644 current/tdp-postgresql/tdp-postgresql.yaml
 create mode 100644 current/tdp-postgresql/values.yaml
 create mode 100644 current/tdp-trino/tdp-trino.yaml
 create mode 100644 current/tdp-trino/values.yaml
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (13/13), 5.34 KiB | 140.00 KiB/s, done.
Total 13 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: . Processing 1 references
remote: Processed 1 references in total
To http://localhost:3001/admin/tdp-gitops.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
$
```

Figura 4 - Publicar current/ no repositório Git

Passo 6 – Acompanhar a sincronização

Verificar o status

Terminal input

```
# Application App-of-Apps (namespace argocd)
kubectl get application -n argocd

# Applications criadas (namespace dos componentes TDP)
kubectl get application -n <NAMESPACE>

# Pods no namespace dos componentes
kubectl get pods -n <NAMESPACE>

# Senha inicial do admin Argo CD
kubectl -n argocd get secret argocd-initial-admin-secret \
  -o jsonpath='{.data.password}' | base64 -d && echo
```

© Copyright Tecnisys 2026. All rights reserved.

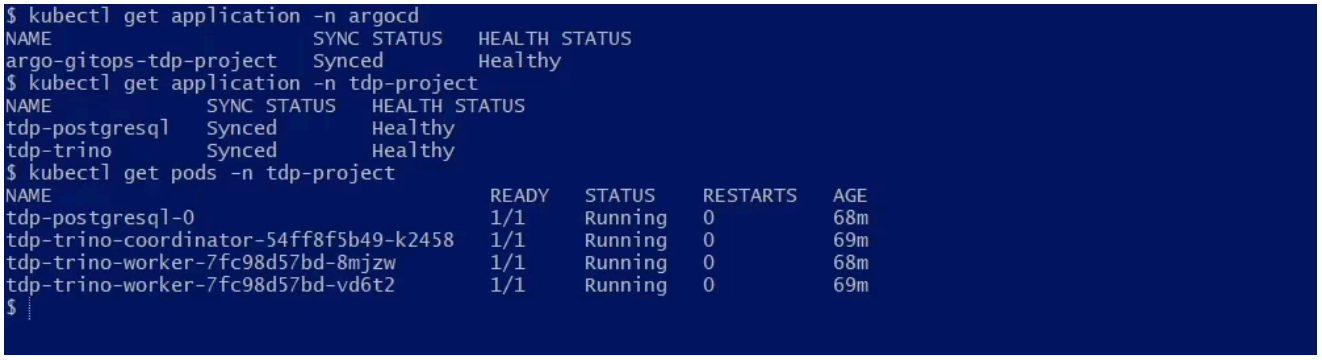


Figura 5 - Acompanhar o status das Applications

Via interface web

Acesse a interface do Argo CD:

Terminal input

```
kubectl get ingress -n argocd
# ou
kubectl get httproute -n argocd
```

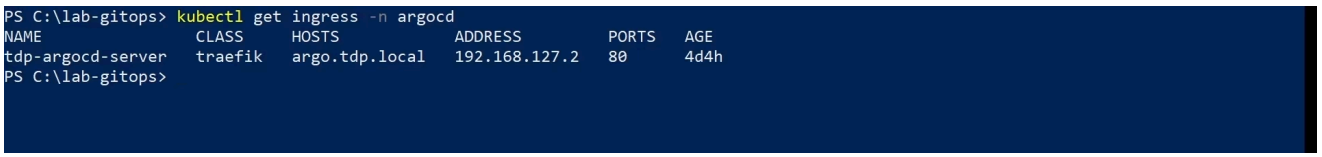


Figura 6 - Descobrir o host exposto

Navegue para a Application configurada em `TDP_APPLICATIONS` e acompanhe o progresso.

Entender o manifest de Application

Cada componente usa um template parametrizado. Exemplo (renderizado):

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: tdp-postgresql
  namespace: <NAMESPACE>
  labels:
    tecnisys.io/cluster: tdp
  finalizers:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    - resources-finalizer.argocd.argoproj.io
spec:
  project: <NAMESPACE>
  source:
    repoURL: registry.tecnisys.com.br/tdp/charts
    targetRevision: <TDP_K8S_RELEASE>
    chart: tdp-postgresql
    helm:
      valueFiles:
        - values.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: <NAMESPACE>
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
      - RespectIgnoreDifferences=true
  ignoreDifferences:
    - group: ""
      kind: Secret
      jqPathExpressions:
        - ".data"
```

Campos importantes:

- `metadata.namespace` — namespace onde a Application é criada (`TDP_PROJECT_NAMESPACE`)
- `spec.project` — AppProject que define as permissões (`TDP_NAMESPACE`)
- `spec.source.repoURL` — registry OCI dos charts TDP
- `spec.source.chart` — nome do chart
- `spec.destination.namespace` — namespace onde os pods são criados (`TDP_NAMESPACE`)
- `syncPolicy.automated` — sincronização automática
- `ignoreDifferences` — ignora mudanças em Secrets (evita *flapping* de senhas)

Customização de componentes

Cada componente tem um `values.yaml` em `available/<componente>/`:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
# Editar valores do componente
vim available/tdp-postgresql/values.yaml

# Rerenderizar
./deploy.sh -v variables.env.local -p

# Commitar e publicar
git add current/tdp-postgresql/
git commit -m "chore: update postgresql values"
git push origin main
```

deploy.sh: referência completa

Terminal input

```
./deploy.sh [OPÇÕES] [COMPONENTES...]
```

Flag	Descrição
`--install`	Primeira instalação: helm login → `tdp-crds` → `tdp-argo` → aguarda *ready* → renderiza → aplica `common`
`-v FILE`	Usar arquivo de variáveis customizado (padrão: `variables.env`)
`-p`	Renderizar `available/` → `current/` apenas (sem aplicar)
`-c`	Aplicar apenas os recursos comuns (`current/common/`) via kubectl
`-a`	Renderizar + aplicar componentes via kubectl (não aplica `common`)

© Copyright Tecnisys 2026. All rights reserved.

Flag	Descrição
`-h`	Exibir ajuda

Exemplos:

Terminal input

```
# Primeira instalação completa (instala Argo CD + bootstrap)
./deploy.sh --install -v variables.env.local

# Re-renderizar templates e publicar (modo GitOps)
./deploy.sh -p -v variables.env.local
git add current/ && git commit -m "chore: render" && git push

# Re-aplicar apenas os recursos comuns (AppProject, Secrets, App of Apps)
./deploy.sh -c -v variables.env.local

# Deploy direto de um componente (sem git push)
./deploy.sh -a tdp-postgresql -v variables.env.local

# Deploy direto de múltiplos componentes
./deploy.sh -a tdp-postgresql tdp-trino -v variables.env.local
```

Verificação da Instalação

Verificação da instalação

Comandos

Verificar o status das Applications

Terminal input

```
argocd app list
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> argocd app list
NAME                  CLUSTER                         NAMESPACE    PROJECT  STATUS  HEALTH   SYNCPOLICY  CONDITIONS  REPO
                                                PATH  TARGET
argocd/tdp-postgresql  https://kubernetes.default.svc  tdp-project  default  Synced  Healthy  Auto-Prune  <none>      registry.tecnisys.com.br/tdp
/charts                                         3.0.0
argocd/tdp-stack       https://kubernetes.default.svc  argocd       default  Synced  Healthy  Auto-Prune  <none>      http://gitea-http.gitea.svc.
cluster.local:3000/admin/tdp-gitops-docs.git  apps  main
PS C:\lab-gitops>
```

Figura 1 - Applications do TDP e seus status

Verificar uma Application específica

Terminal input

```
argocd app get tdp-postgresql
```

```
PS C:\lab-gitops> argocd app get tdp-postgresql
Name:               argocd/tdp-postgresql
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          tdp-project
URL:                https://argo.tdp.local/applications/tdp-postgresql
Source:
- Repo:             registry.engtecnisys.com.br/tdp/charts
  Target:           0.1.0
SyncWindow:         Sync Allowed
Sync Policy:        Automated (Prune)
Sync Status:        Synced to 0.1.0
Health Status:      Healthy

GROUP              KIND                   NAMESPACE    NAME               STATUS  HEALTH   HOOK  MESSAGE
                   Secret                 tdp-project  tdp-postgresql     Synced                 secret/tdp-postgresql configured
                   Service                tdp-project  tdp-postgresql     Synced  Healthy
                   Service                tdp-project  tdp-postgresql-hl  Synced  Healthy
                   ServiceAccount         tdp-project  tdp-postgresql     Synced
apps               StatefulSet            tdp-project  tdp-postgresql     Synced  Healthy
networking.k8s.io  NetworkPolicy          tdp-project  tdp-postgresql     Synced
policy             PodDisruptionBudget    tdp-project  tdp-postgresql     Synced  Healthy
PS C:\lab-gitops>
```

Figura 2 - Verificar a Application postgresql

Verificar os Pods no cluster

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

```
PS C:\lab-gitops> kubectl get pods -n tdp-project
NAME               READY   STATUS    RESTARTS      AGE
tdp-postgresql-0   1/1     Running   2 (62m ago)   3h5m
PS C:\lab-gitops>
```

Figura 3 - Verificar os pods

Todos os pods devem apresentar status `Running` com todas as réplicas prontas.

© Copyright Tecnisys 2026. All rights reserved.

Verificar os Serviços

Terminal input

```
kubectl get svc -n <NAMESPACE>
```

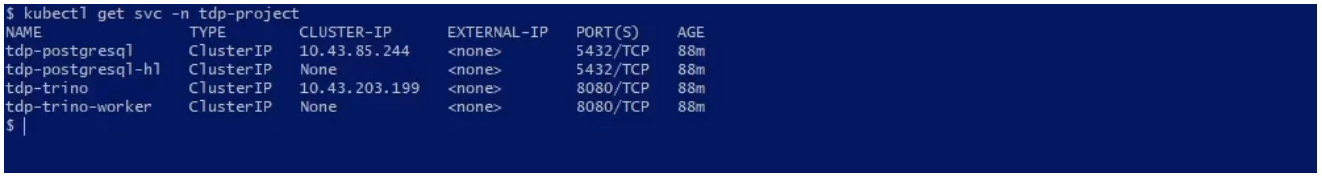


Figura 4 - Verificar os Serviços

Verificar os Volumes Persistentes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

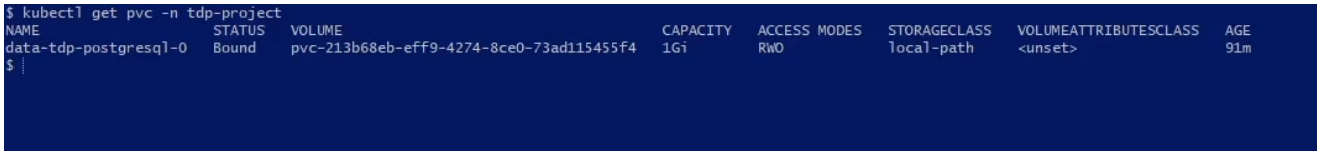


Figura 5 - Verificar os Volumes Persistentes

Todos os *Persistent Volume Claims* devem apresentar status `Bound`.

Resolução de Problemas

Resolução de problemas

Application em OutOfSync ou Degraded

Se uma Application aparece com status `OutOfSync` ou `Degraded`:

1. Verifique os detalhes da Application:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
argocd app get <componente>
```

2. Verifique os eventos e logs dos pods:

Terminal input

```
kubectl describe pod <pod-name> -n <NAMESPACE>
kubectl logs <pod-name> -n <NAMESPACE>
```

3. Tente sincronizar manualmente:

Terminal input

```
argocd app sync <componente>
```

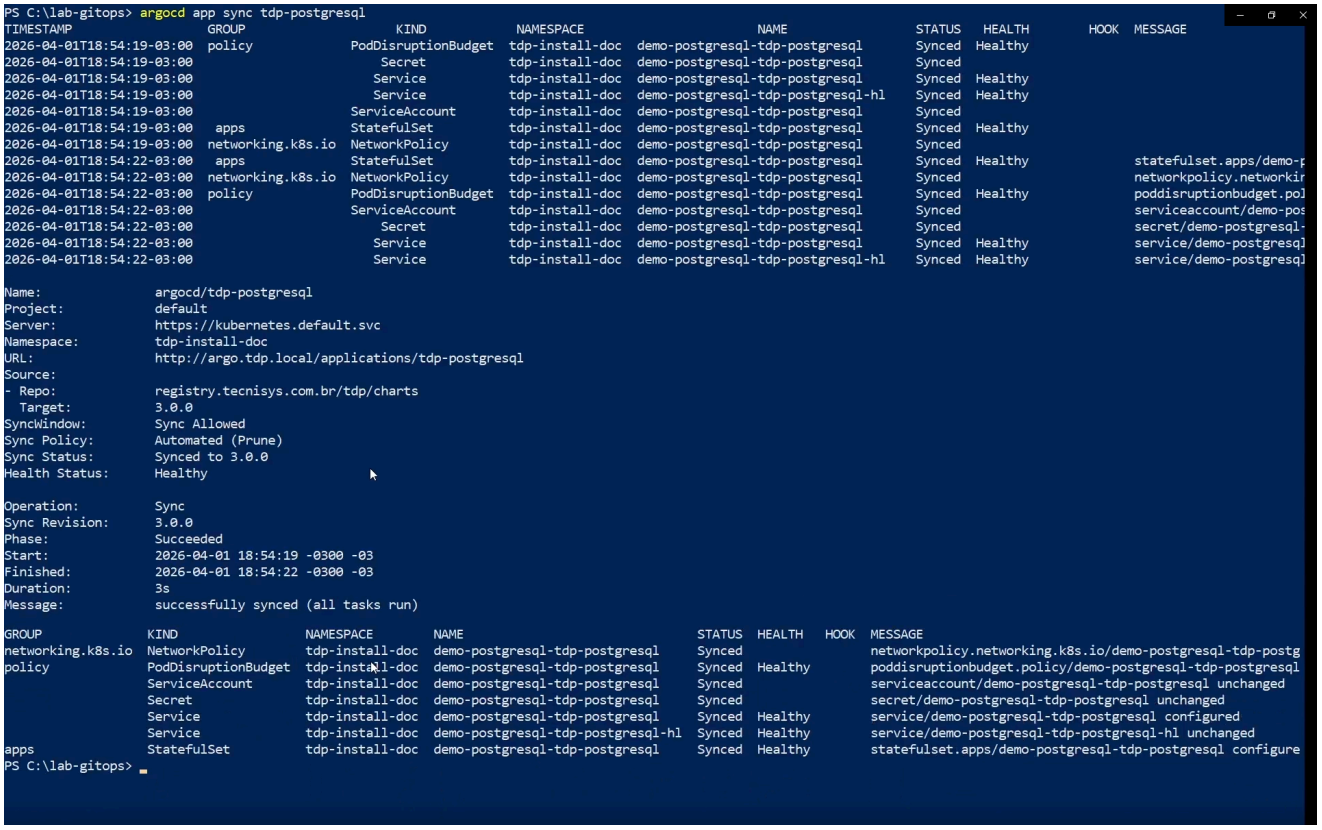


Figura 1 - app sync

© Copyright Tecnisys 2026. All rights reserved.



Erro de acesso ao registry OCI

Se o Argo CD apresenta erro ao tentar baixar um chart:

1. Verifique se o secret do registry foi aplicado:

Terminal input

```
kubectl get secret tdp-helm-oci-tdp -n argocd -o yaml
```

2. As credenciais podem ter expirado. Reaplique os recursos comuns:

Terminal input

```
./deploy.sh -c -v variables.env.local
```

3. Verifique a conectividade do cluster com o registry:

Terminal input

```
kubectl run test-dns --rm -it --image=busybox -n <NAMESPACE> -- \
  sh -c "nslookup registry.tecnisys.com.br"
```

Componentes com dependência do PostgreSQL falhando

Se Airflow, Superset, Ranger, OpenMetadata ou CloudBeaver apresentam falhas:

1. Verifique se o PostgreSQL está operacional:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl get pods -n <NAMESPACE> | grep tdp-postgresql
```

2. Se o PostgreSQL ainda não está `Running`, aguarde. O `selfHeal` do Argo CD reconciliará os componentes dependentes automaticamente.

3. Para forçar a reconciliação manualmente:

Terminal input

```
argocd app sync tdp-airflow
```

Application em namespace errado

As Applications devem estar no namespace `<NAMESPACE>` (definido em `TDP_PROJECT_NAMESPACE`), **não** em `argocd`. Confirme que o `argocd-cm` tem `application.namespaces` configurado para o namespace do projeto (ver Passo 3).

Debug de sync

Terminal input

```
# Ver eventos da Application
kubectl describe application <componente> -n <NAMESPACE>

# Logs do application controller
kubectl logs -n argocd -l app.kubernetes.io/name=argocd-application-controller --tail=100

# Forçar reconciliação
kubectl annotate application <componente> -n <NAMESPACE> \
  argocd.argoproj.io/refresh=hard --overwrite
```

Rollback de uma Application

O Argo CD mantém histórico de sincronizações. Para reverter:

- **Via interface web:** selecione a Application → **History and Rollback** → escolha a revisão → **Rollback**.

© Copyright Tecnisys 2026. All rights reserved.

- Via CLI:

```
# Listar o histórico
argocd app history <componente>

# Reverter para uma revisão específica
argocd app rollback <componente> <revision-id>
```

> **WARNING**
>
> Ao realizar rollback de um componente que depende do PostgreSQL, verifique se a versão do banco de dados é compatível com a versão do componente sendo revertido.

Atualizar Versão dos Charts

Atualizar versão dos charts

```
# 1. Atualizar HELM_CHART_VERSION em variables.env.local
vim variables.env.local

# 2. Rerenderizar
./deploy.sh -v variables.env.local -p

# 3. Commitar
git add current/
git commit -m "chore: bump chart version"
git push origin main
```

Após o push, o Argo CD detecta a mudança em `current/` e atualiza os componentes automaticamente.

Compatibilidade ArgoCD

© Copyright Tecnisys 2026. All rights reserved.



Todos os charts TDP foram ajustados para compatibilidade total com o Argo CD:

- **Single-source:** as Applications usam apenas a fonte OCI (sem *multi-source*).
- **ignoreDifferences:** Secrets ignorados para evitar *flapping* de senhas.
- **Check-or-reuse:** credenciais reutilizadas quando já existem (sem rotação em *sync*).
- **Namespace separado:** Applications em `TDP_PROJECT_NAMESPACE`.
- **Sync waves:** ordenação correta de recursos dependentes.

Componentes com ajustes específicos para Argo CD:

- **tdp-jupyter:** correção da rotação de senha no `db-create-job`
- **tdp-trino:** remoção de *hooks* deletáveis (PVC/Job)
- **tdp-hive-metastore:** correção da rotação de senha + *security context*
- **tdp-superset:** correção do `randAlphaNum` do PostgreSQL (Bitnami)
- **tdp-clickhouse:** *cleanup job* desabilitado por padrão
- **tdp-airflow:** remoção de *lookup* em secrets

Segurança

- `variables.env.local` está no `.gitignore` — **nunca** commite tokens ou senhas.
- Use **Kubernetes Secrets** para dados sensíveis em runtime.
- Rotacione os tokens do git e do registry regularmente.
- O `TDP_PROJECT_NAMESPACE` deve ter **RBAC restrito**.
- Os Secrets do registry e do git são versionados como **templates** em `common/` (preenchidos a partir de `variables.env.local`).

Componentes disponíveis

Componente	Descrição	Dependências
`tdp-postgresql`	PostgreSQL	—
`tdp-trino`	Trino SQL engine	—
`tdp-airflow`	Apache Airflow	PostgreSQL
`tdp-clickhouse`	ClickHouse	—

© Copyright Tecnisys 2026. All rights reserved.

Componente	Descrição	Dependências
`tdp-cloudbeaver`	CloudBeaver UI	PostgreSQL
`tdp-deltalake`	Delta Lake	—
`tdp-hive-metastore`	Hive Metastore	PostgreSQL
`tdp-iceberg`	Apache Iceberg	—
`tdp-jupyter`	JupyterHub	—
`tdp-kafka`	Apache Kafka (Strimzi)	—
`tdp-nifi`	Apache NiFi	—
`tdp-openmetadata`	OpenMetadata	PostgreSQL
`tdp-ozone`	Apache Ozone S3	—
`tdp-ranger`	Apache Ranger	—
`tdp-spark`	Apache Spark	—
`tdp-superset`	Apache Superset	PostgreSQL

Footnotes

1. OCI (Open Container Initiative) é o padrão moderno para armazenar não apenas imagens de contêiner, mas também pacotes Helm. Por isso, ao cadastrar o registry, é necessário marcar a opção **Enable OCI** — sem ela, o Argo CD tentará tratar o endereço como um repositório Helm convencional baseado em `index.yaml` e a conexão falhará. ↩

© Copyright Tecnisys 2026. All rights reserved.

Políticas de Cluster

Políticas de cluster são manifests que controlam a comunicação entre Pods, entre namespaces e entre o cluster e redes externas. Elas não instalam componentes — definem quais conexões são permitidas ou bloqueadas no nível da plataforma.

No contexto da instalação do TDP Kubernetes, essas políticas só são necessárias quando o ambiente impõe restrições de conectividade que afetam o funcionamento dos componentes.

Um conceito recorrente nessas políticas é o **egress**: o tráfego de saída de um Pod em direção a serviços externos ao próprio workload — como bancos de dados, endpoints S3, APIs corporativas e object storage. Ambientes com NetworkPolicies ativas bloqueiam esse tráfego por padrão, exigindo que cada destino seja declarado explicitamente.

Quando considerar este ajuste

Considere este ajuste quando o cluster adotar **NetworkPolicies** ativas ou outras regras de conectividade que possam bloquear a comunicação necessária aos componentes do TDP.

Dependendo do ambiente, essas políticas podem precisar ser aplicadas antes da instalação dos componentes ou em conjunto com ela.

Sem essas regras, componentes do TDP podem falhar ao tentar acessar:

- bancos de dados externos
- endpoints S3 e object storage
- serviços corporativos fora do namespace
- outros intervalos de IP liberados pela rede do ambiente

Papel deste YAML no fluxo GitOps

O YAML abaixo é um manifest de política de plataforma. Ele não instala componentes do TDP e não substitui os manifests das Applications. Seu papel é liberar ou restringir comunicações exigidas pelo ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Exemplo de NetworkPolicy

O exemplo abaixo permite comunicação interna entre Pods do namespace e libera egress para destinos externos específicos:

allow-internal-full-and-external-ip.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-internal-full-and-external-ip
  namespace: <NAMESPACE>
spec:
  podSelector: {}
  policyTypes:
    - Egress
    - Ingress

  ingress:
    - from:
        - podSelector: {}

  egress:
    - to:
        - podSelector: {}
    - to:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system

    - to:
        - ipBlock:
            cidr: x.x.x.0/24
```

Aplicação

Terminal input

```
kubectl apply -f allow-internal-full-and-external-ip.yaml
```

O que revisar antes de aplicar

© Copyright Tecnisys 2026. All rights reserved.



- O namespace de destino (`<NAMESPACE>`)
- Os intervalos de IP liberados em `ipBlock.cidr`
- A necessidade de permitir comunicação com `kube-system`
- As políticas já existentes no cluster, para evitar sobreposição ou bloqueios involuntários

Onde guardar no repositório GitOps

Para manter rastreabilidade, versione este arquivo no repositório GitOps dentro de uma pasta `policies/`:

```
tdp-gitops/
├── app-of-apps.yaml
├── apps/
├── values/
└── policies/
    └── allow-internal-full-and-external-ip.yaml
```

Esses arquivos não são gerenciados pelo Argo CD como Applications — são aplicados diretamente com `kubectl apply -f`. Armazená-los no repositório garante histórico de mudanças junto ao restante da configuração da plataforma.

© Copyright Tecnisys 2026. All rights reserved.

Registry Privado com Kyverno

Clusters que baixam imagens de um registry privado precisam fornecer credenciais de acesso a cada workload.
Em um ambiente com muitos componentes — como o TDP Kubernetes — repetir essa configuração manualmente em cada chart é ineficiente e gera riscos de inconsistência.

O **Kyverno** resolve esse problema como um policy engine de cluster: ele intercepta a criação de Pods e injeta automaticamente o `imagePullSecrets` necessário, com base em uma `ClusterPolicy` centralizada.
Dessa forma, todos os componentes passam a usar as credenciais do registry sem que cada manifest precise declará-las individualmente.

Esse ajuste é condicional: só se aplica quando o ambiente exige autenticação para baixar imagens de um registry privado.
Ambientes que usam registries públicos ou que configuram `imagePullSecrets` diretamente em cada chart não precisam desta etapa.

Quando considerar este ajuste

Considere esta configuração quando:

- as imagens são distribuídas por um registry privado
- o cluster precisa reutilizar as mesmas credenciais em múltiplos componentes
- a política de plataforma exige padronização do uso de `imagePullSecrets`
- se deseja centralizar a manutenção dessas credenciais

Papel do Kyverno neste contexto

O Secret continua sendo o objeto que armazena as credenciais do registry.
O papel do Kyverno é aplicar e padronizar o uso desse Secret no cluster por meio de uma `ClusterPolicy`.
Assim, novos Pods criados nos namespaces configurados podem receber automaticamente a referência ao `imagePullSecrets`.

> ⚠ IMAGEPULLSECRETS E NAMESPACE

© Copyright Tecnisys 2026. All rights reserved.

O Kubernetes só utiliza um `imagePullSecrets` se o Secret referenciado **existir no mesmo namespace** do Pod.
Se o Secret `tdp-registry` não existir no namespace onde o Pod será criado, o download da imagem falhará mesmo com a `ClusterPolicy` ativa.
Crie o Secret em cada namespace TDP antes de criar os Pods nesse namespace.

MUTAÇÃO AFETA APENAS NOVOS PODS

A `ClusterPolicy` do Kyverno atua no momento da criação de Pods.
Workloads já existentes não são afetados — apenas novos Pods criados após a aplicação da policy receberão o `imagePullSecrets`.
Para aplicar as credenciais a workloads existentes, é necessário recriá-los (ex.: `kubectl rollout restart deployment/<NAME> -n <NAMESPACE>`).

Visão geral da configuração

O fluxo é composto por três etapas:

1. Instalar o Kyverno no cluster
2. Criar o Secret do registry **em cada namespace TDP**
3. Criar uma `ClusterPolicy` que injeta `imagePullSecrets` automaticamente nos Pods

1. Instalar o Kyverno

Terminal input

```
helm repo add kyverno https://kyverno.github.io/kyverno/
helm repo update
helm install kyverno kyverno/kyverno --namespace kyverno --create-namespace
```

2. Criar o Secret do registry em cada namespace

NÃO VERSIONE O SECRET COM CREDENCIAIS NO GIT

O arquivo de Secret contém credenciais de acesso ao registry em texto simples.

© Copyright Tecnisys 2026. All rights reserved.



> **Nunca versione esse arquivo no repositório Git.**
> Gerencie as credenciais via ferramenta especializada (Sealed Secrets, External Secrets Operator, HashiCorp Vault) ou crie o Secret manualmente em cada namespace.

Crie o Secret com as credenciais do registry em **cada namespace TDP** onde os componentes serão instalados:

Terminal input

```
kubectl create secret docker-registry tdp-registry \
  --docker-server=registry.tecnisys.com.br \
  --docker-username='<USERNAME>' \
  --docker-password='<PASSWORD>' \
  -n <NAMESPACE>
```

Repita o comando para cada namespace TDP (ex.: `tdp`, `tdp-kafka`, `tdp-airflow`, etc.) ou automatize com um script de bootstrap.

> **ALTERNATIVA: SINCRONIZAR VIA KYVERNO CLONESECRETPOLICY**
> O Kyverno também suporta a cópia automática de Secrets entre namespaces por meio de uma `ClusterPolicy` do tipo `clone`.
> Com essa abordagem, basta criar o Secret em um namespace de origem (ex.: `kyverno`) e o Kyverno o replica automaticamente nos namespaces de destino.
> Consulte a documentação do Kyverno para detalhes sobre `generate` e `clone`.

3. Criar a policy para injetar `imagePullSecrets`

Crie um arquivo `policy-kyverno.yaml`:

policy-kyverno.yaml

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: inject-tecnisys-registry-secret
```

© Copyright Tecnisys 2026. All rights reserved.

```
  annotations:
    policies.kyverno.io/title: Inject Image Pull Secrets
    policies.kyverno.io/description: Automatically adds the Tecnisys secret to
all pods.
spec:
  background: false
  rules:
    - name: inject-secret
      match:
        any:
          - resources:
              kinds:
                - Pod
      mutate:
        patchStrategicMerge:
          spec:
            imagePullSecrets:
              - name: tdp-registry
```

Aplique a policy:

Terminal input

```
kubectl apply -f policy-kyverno.yaml
```

Resultado esperado

Após a aplicação da `ClusterPolicy`, novos Pods criados no cluster passam a receber o `imagePullSecrets` com o nome `tdp-registry`, permitindo o download das imagens privadas do registry da Tecnisys.

O Secret precisa existir no namespace do Pod para que o Kubernetes o utilize — confirme a criação do Secret em todos os namespaces TDP antes de instalar os componentes.

Onde guardar no repositório GitOps

Versione **apenas** o arquivo `policy-kyverno.yaml` no repositório GitOps dentro de uma pasta `policies/`:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-gitops/
├── app-of-apps.yaml
├── apps/
├── values/
└── policies/
    └── kyverno-registry.yaml
```

Não versione o arquivo `tdp-registry.yaml` com as credenciais.
Esses arquivos de policy são aplicados com `kubectl apply -f`, como parte do bootstrap do ambiente — não são gerenciados diretamente pelo Argo CD como Applications.

> **BOOTSTRAP VS. ARGO CD**
>
> Os manifests da pasta `policies/` são aplicados uma vez, durante o bootstrap do ambiente.
> Eles não precisam ser gerenciados por uma Application do Argo CD, mas podem ser se houver uma Application de plataforma responsável pela gestão de políticas do cluster.

© Copyright Tecnisys 2026. All rights reserved.

CONFIGURAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Configuração Geral

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Configuração de componentes TDP

A configuração é a etapa em que um componente TDP Kubernetes é adaptado ao ambiente e ao modo de operação adotado.
Ela define como o componente será implantado, quais recursos utilizará, como será exposto, de quais dependências precisa e como se integrará aos demais serviços da plataforma.

Na prática, a configuração permite ajustar parâmetros como namespace, CPU, memória, armazenamento, secrets, acesso externo, persistência e integrações compartilhadas — como banco de dados, object storage e serviços de apoio.
Com isso, o mesmo componente pode ser usado em contextos diferentes, como laboratório, homologação e produção, sem mudar sua função principal.

Por exemplo:

- Conectar o Airflow ao PostgreSQL do cluster em vez de usar o banco interno
- Dizer ao Kafka quantas réplicas de broker e qual fator de replicação mínimo usar
- Informar ao NiFi onde está o servidor LDAP para autenticação
- Configurar o S3 Gateway do Ozone com as credenciais certas para que o Spark possa gravar dados

Os parâmetros de configuração se aplicam independentemente do método de instalação utilizado.
O objetivo desta página é explicar o que normalmente pode ser configurado em um componente TDP e como esses ajustes funcionam na prática.

Ao longo desta seção, são apresentados os padrões de configuração mais comuns entre os componentes TDP:

© Copyright Tecnisys 2026. All rights reserved.

Área	O que define	Exemplos
Execução no cluster	Onde e com quais recursos o componente será executado.	Namespace, CPU, memória, réplicas e limites.
Persistência	Como os dados serão armazenados.	Volumes, disco, storage class e retenção.
Exposição do serviço	Como o componente será acessado.	Service, porta, ingress, hostname e TLS.
Segurança e acesso	Como credenciais e dados sensíveis serão fornecidos.	Secrets, usuários, senhas, tokens e referências a secrets.
Integrações	Como o componente se conecta a outros serviços.	PostgreSQL, S3/Ozone, endpoints e serviços compartilhados.
Forma de aplicação	Como a configuração chega ao cluster.	Helm, Argo CD/GitOps e futuramente tdpctl.

Onde consultar os parâmetros disponíveis

Antes de personalizar a configuração de um componente, existem três fontes principais:

- **A página do componente** neste guia — cada componente tem uma página com os parâmetros suportados, exemplos e observações específicas.
- **O `values.yaml` do chart** — contém todos os parâmetros aceitos com seus valores padrão e comentários explicativos; acesse com `helm show values`.
- **O README do chart** — disponível no registry, oferece uma visão geral das opções e casos de uso.

> **INFO**

© Copyright Tecnisys 2026. All rights reserved.

Execute `helm show values oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME>` para exportar todos os parâmetros disponíveis com seus valores padrão e comentários explicativos.

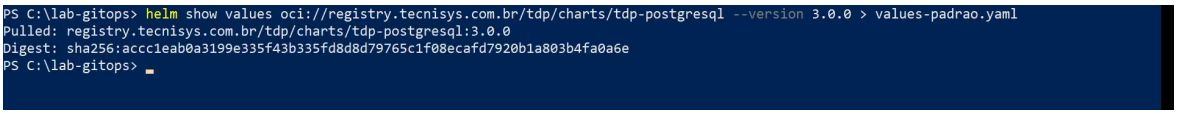


Figura 1 — Exportar os valores padrão do chart

Este comando exporta os valores padrão do chart para um arquivo local `values-padrao.yaml`.
Esse arquivo pode ser usado como referência para identificar quais parâmetros ajustar para o ambiente.

Arquivo de valores customizado

A configuração de um componente é feita por meio de um arquivo YAML com apenas os parâmetros que precisam ser alterados em relação ao padrão do chart.
Não é necessário copiar o arquivo completo de valores do chart — inclua somente as chaves que deseja sobrescrever.

O arquivo de valores define **o que** será configurado. Helm ou Argo CD definem **como** essa configuração será aplicada ao cluster — em ambos os fluxos, o formato e os parâmetros são idênticos.

Exemplo mínimo de arquivo de valores:

meu-values.yaml

```yaml
# Inclua apenas o que precisa ser diferente do padrão
resources:
  requests:
    cpu: "500m"
    memory: "1Gi"
  limits:
    cpu: "2"
    memory: "2Gi"

persistence:
```

© Copyright Tecnisys 2026. All rights reserved.

```
  size: 20Gi
  storageClassName: "nfs-fast"
```

Como aplicar a configuração

Os procedimentos a seguir mostram como aplicar os valores ao cluster via Helm ou Argo CD.

Via Helm

Comandos

> **NOTE**
>
> O Helm mescla automaticamente seus valores com os padrões do chart — inclua apenas as chaves que deseja customizar.

Formas de fornecer valores

Arquivo de valores (recomendado) — crie um YAML com as chaves a alterar e passe com `-f`:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace \
  -f meu-values.yaml
```

Flag `--set` — para alterações pontuais diretamente na linha de comando:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace \
  --set parametro.chave=valor
```

© Copyright Tecnisys 2026. All rights reserved.

Combinação de arquivos — múltiplos arquivos mesclados em ordem, com os últimos tendo precedência:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace \
  -f values-base.yaml \
  -f values-producao.yaml
```

Passo a passo

> **DESCOBRIR PARÂMETROS DISPONÍVEIS**
>
> Antes de começar, exporte os valores aceitos pelo chart. Consulte Onde consultar os parâmetros disponíveis.

1. Crie o arquivo de valores (exemplo: `meu-values.yaml`):

PowerShell

```
code .\meu-values.yaml
```

2. Ajuste os parâmetros desejados, como namespace, recursos ou integrações:

meu-values.yaml

```
resources:
  requests:
    cpu: "500m"
    memory: "1Gi"
  limits:
    cpu: "1"
    memory: "2Gi"
```

3. Aplique a configuração com o arquivo criado:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace \
  -f meu-values.yaml
```

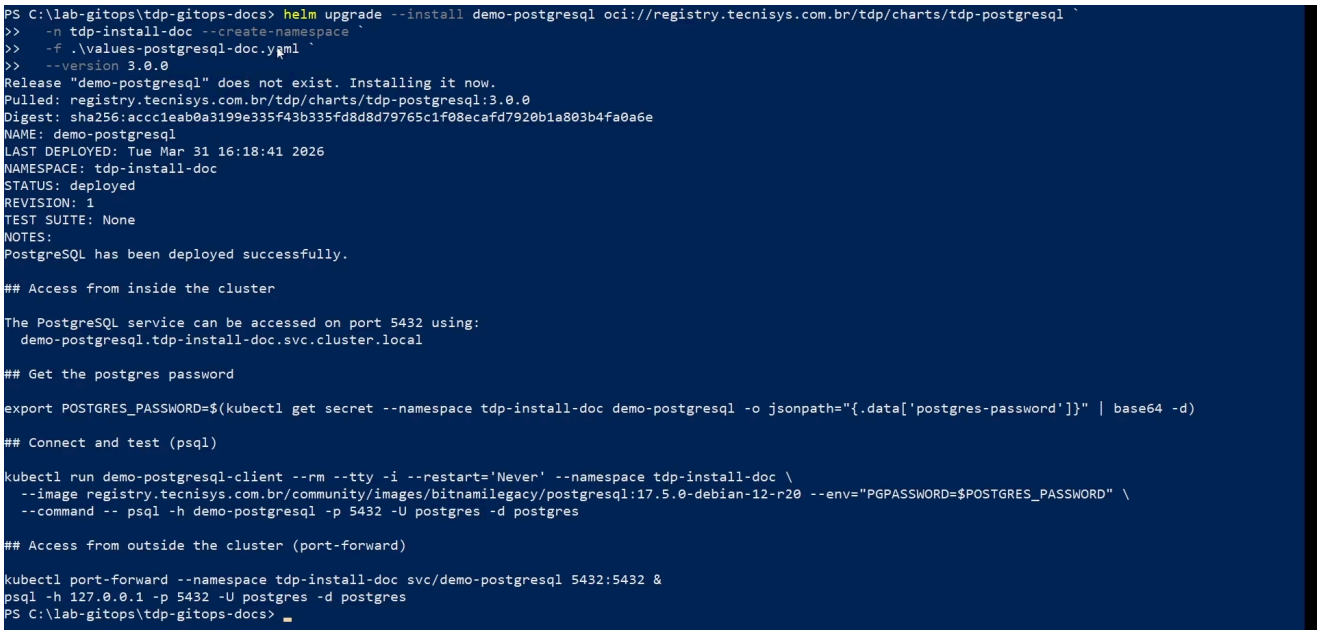


Figura 3 - Configuração do arquivo criado

4. Verifique se os valores configurados foram aplicados:

Terminal input

```
helm get values <RELEASE_NAME> -n <NAMESPACE>
```

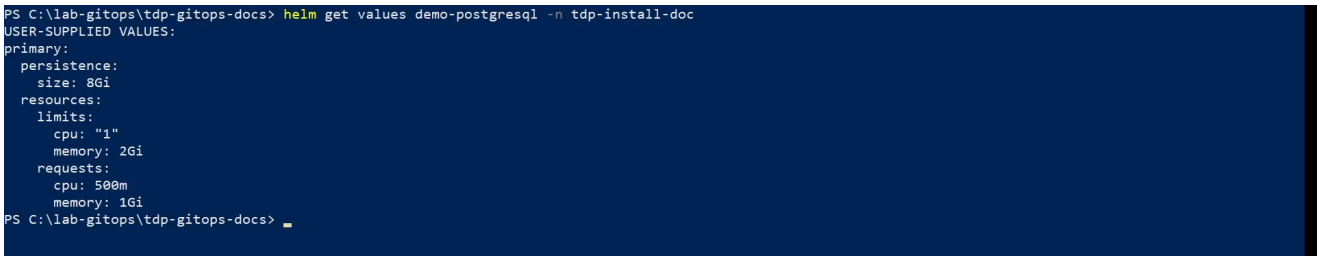


Figura 4 - Verificação dos valores configurados

5. Verifique os pods:

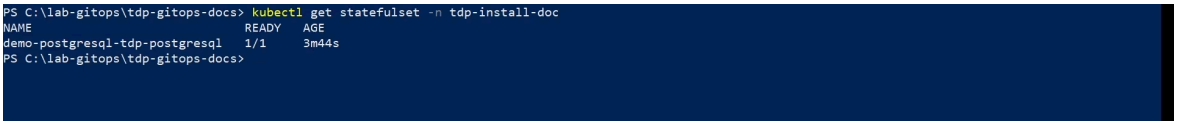
© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl get pods -n <NAMESPACE>
```



Figura 5 - Verificação dos pods

> **NOTE**
>
> Para componentes stateful, como PostgreSQL, valide também:
>
> ```
> kubectl get statefulset -n <NAMESPACE>
> ```
>
> 
>
>
> *Figura 6 - Validação de componentes stateful*
>
> Para componentes que utilizam Deployment, valide também:
>
> ```
> kubectl get deploy -n <NAMESPACE>
> ```

6. Confirme que o release está com status `deployed` e com a revisão atualizada:

Terminal input

```
helm list -n <NAMESPACE>
```

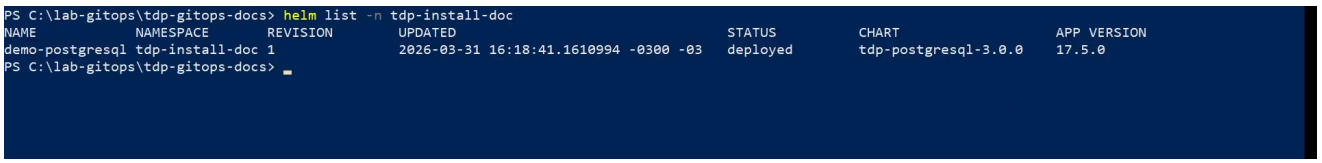


Figura 7 - Helm list

© Copyright Tecnisys 2026. All rights reserved.

Declaração via GitOps com Argo CD

Comandos

Como funciona

No fluxo GitOps, a configuração do componente é declarada em arquivos `values.yaml` versionados no repositório Git — exatamente os mesmos parâmetros usados no fluxo Helm.
Não há um conjunto separado de parâmetros "do Argo CD": o que muda é que o Argo CD monitora o repositório continuamente, compara o estado declarado no Git com o estado atual do cluster e aplica as diferenças — sem necessidade de rodar comandos de instalação diretamente.

Três conceitos centrais orientam esse fluxo:

Application — recurso Kubernetes gerenciado pelo Argo CD que aponta para um chart Helm e um arquivo de valores no repositório.
Cada componente TDP tem sua própria Application.

Sincronização (Sync) — processo pelo qual o Argo CD compara o estado declarado no Git com o estado atual do cluster e aplica as diferenças.
Pode ser automática (quando `automated` está habilitado na Application) ou manual (`argocd app sync`).

Reconciliação contínua — quando `selfHeal: true` está configurado, o Argo CD corrige automaticamente qualquer desvio entre o cluster e o Git, mesmo que alguém tenha alterado algo diretamente no cluster.

Para alterar a configuração de um componente, o arquivo de valores é editado no repositório e a alteração é enviada ao Git.
O Argo CD detecta a mudança e aplica — sem necessidade de rodar comandos de instalação.

Passo a passo

1. Acesse o repositório GitOps:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
cd ./tdp-GitOps
```

2. Abra o arquivo de valores do componente indicado na página específica do componente:

PowerShell

```
code .\caminho\do\arquivo.yaml
```

Ou, alternativamente:

PowerShell

```
notepad .\caminho\do\arquivo.yaml
```



Figura 8 - Abre o arquivo de valores

3. Ajuste os parâmetros desejados no arquivo:

de : arquivo.yaml

```
primary:
  resources:
    requests:
      cpu: "500m"
      memory: "512Mi"
persistence:
  size: 10Gi
```

para : arquivo.yaml

```
primary:
  resources:
    requests:
```

© Copyright Tecnisys 2026. All rights reserved.

```
      cpu: "1200m"
      memory: "1Gi"
    limits:
      cpu: "1"
      memory: "2Gi"
persistence:
  size: 25Gi
```

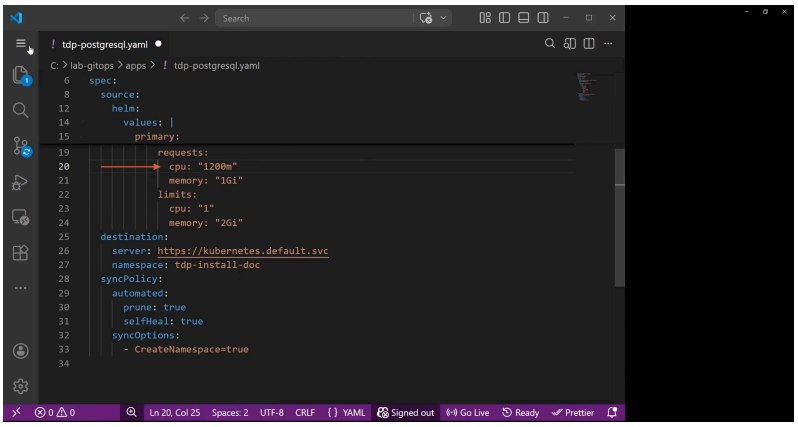


Figura 9 - Alteração dos valores - exemplo

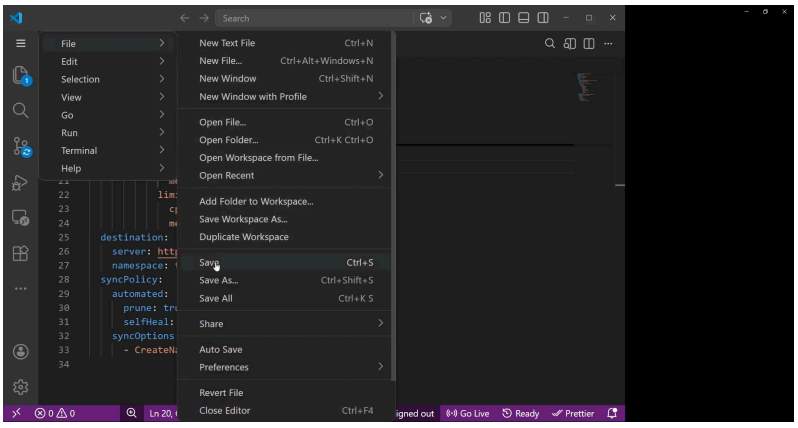


Figura 9a - Alteração dos valores - exemplo

4. Salve e envie a alteração:

Terminal input

```
git add .
git commit -m "config: ajustar parâmetros do componente"
git push origin main
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> git add .
PS C:\lab-gitops> git commit -m "config: ajustar parâmetros do componente"
[main d158839] config: ajustar parâmetros do componente
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\lab-gitops> git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 383 bytes | 383.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: . Processing 1 references
remote: Processed 1 references in total
To http://git.tdp.local/admin/tdp-gitops-docs.git
   5ec71dc..d158839  main -> main
PS C:\lab-gitops>
```

Figura 10 - Salva e envia alteração

O Argo CD detectará automaticamente a alteração e iniciará a sincronização.

5. Verifique se a Application evolui para `Synced` e `Healthy`:

Terminal input

```
argocd app list
```

```
PS C:\lab-gitops> argocd app list
NAME                  CLUSTER                         NAMESPACE    PROJECT  STATUS     HEALTH   SYNCPOLICY  CONDITIONS  REPO
                                                           PATH  TARGET
argocd/tdp-postgresql  https://kubernetes.default.svc  tdp-project  default  OutOfSync  Healthy  Auto-Prune  <none>      registry.engte
cnisys.com.br/tdp/charts                                          3.0.0
argocd/tdp-stack       https://kubernetes.default.svc  argocd       default  OutOfSync  Healthy  Auto-Prune  <none>      http://gitea-h
ttp.gitea.svc.cluster.local:3000/admin/tdp-gitops-docs.git  apps  main
PS C:\lab-gitops>
```

Figura 11 - Verifica Application

6. Para inspecionar a Application do componente em detalhe:

Terminal input

```
argocd app get <APPLICATION_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> argocd app get tdp-postgresql --grpc-web
Name:               argocd/tdp-postgresql
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          tdp-project
URL:                http://argo.tdp.local/applications/tdp-postgresql
Source:
- Repo:             registry.engtecnisys.com.br/tdp/charts
  Target:           3.0.0
SyncWindow:         Sync Allowed
Sync Policy:        Automated (Prune)
Sync Status:        OutOfSync from 3.0.0
Health Status:      Healthy

GROUP              KIND                   NAMESPACE    NAME                STATUS     HEALTH   HOOK  MESSAGE
                   Secret                 tdp-project  tdp-postgresql      OutOfSync                 secret/tdp-postgresql unchanged
                   Service                tdp-project  tdp-postgresql      Synced     Healthy
                   Service                tdp-project  tdp-postgresql-hl   Synced     Healthy
                   ServiceAccount         tdp-project  tdp-postgresql      Synced
apps               StatefulSet            tdp-project  tdp-postgresql      Synced     Healthy
networking.k8s.io  NetworkPolicy          tdp-project  tdp-postgresql      Synced
policy             PodDisruptionBudget    tdp-project  tdp-postgresql      Synced     Healthy
PS C:\lab-gitops>
```

Figura 12 - Inspeciona Application do componente

7. Se a sincronização automática não estiver habilitada, ou se quiser aplicar a mudança imediatamente:

Terminal input

```
argocd app sync <APPLICATION_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.

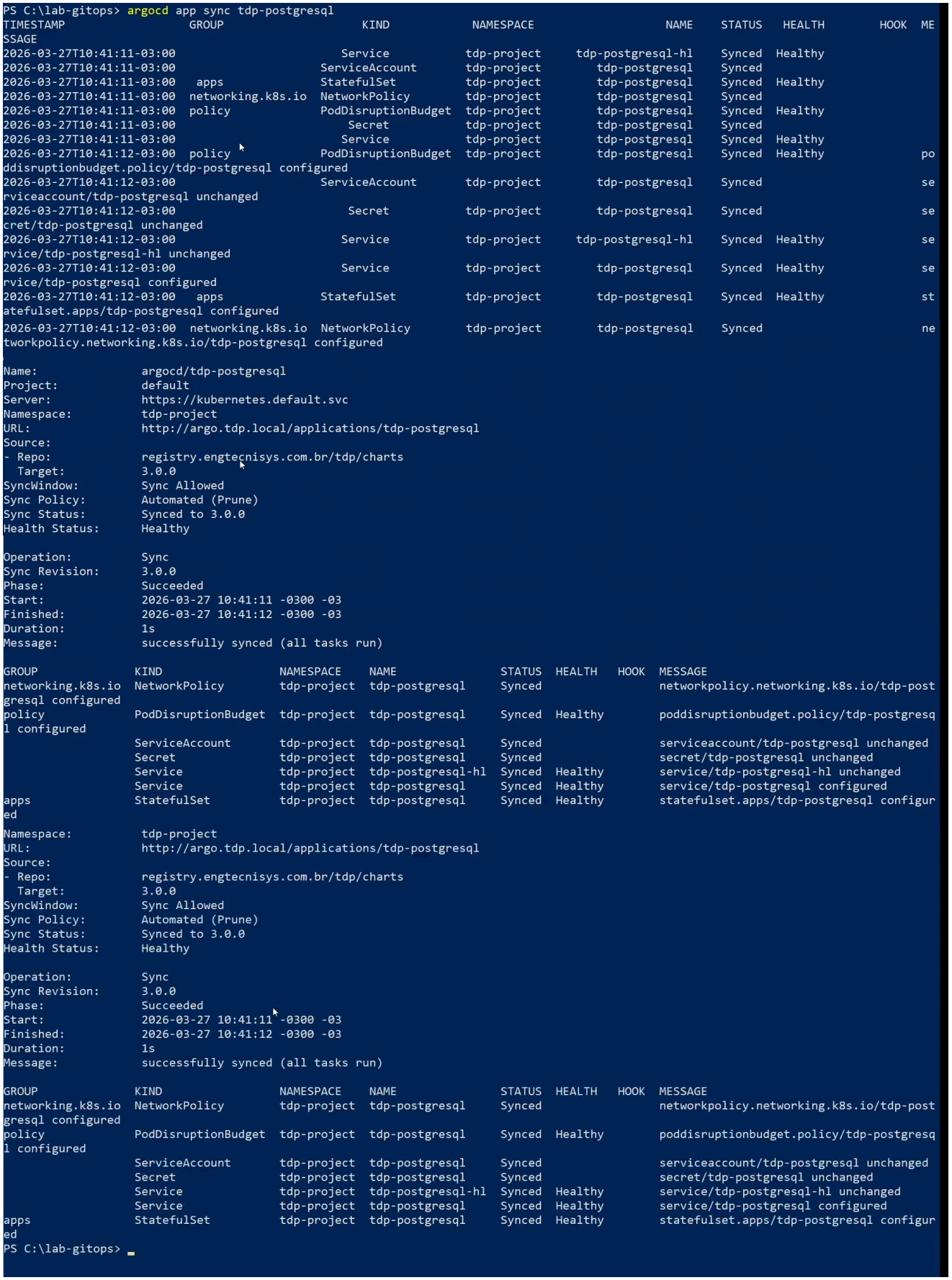


Figura 13 - Sincronização manual

© Copyright Tecnisys 2026. All rights reserved.

8. Verifique os pods:

Terminal input

```
kubectl get pods -n <NAMESPACE>
```



Figura 14 - Verifica os pods

> **NOTE**
>
> Para componentes stateful, como PostgreSQL, valide também:
>
> ```
> kubectl get statefulset -n <NAMESPACE>
> ```
>
> 
>
>
> *Figura 15 - Verifica componentes stateful*
>
> Para componentes que utilizam Deployment, valide também:
>
> ```
> kubectl get deploy -n <NAMESPACE>
> ```
>
> Confirme que os pods estão em `Running` e que os workloads mostram todas as réplicas prontas.

Padrões comuns entre os charts TDP

Os charts do TDP Kubernetes compartilham padrões de configuração descritos a seguir. Consulte o `values.yaml` do chart e a página do componente neste guia para os parâmetros exatos.

© Copyright Tecnisys 2026. All rights reserved.

Padrão do componente e revisão da configuração

Em geral, a primeira instalação de um componente parte do comportamento padrão definido pelo chart.
Esse padrão existe para permitir que o serviço seja implantado com o conjunto mínimo de dependências externas, facilitando validação inicial, testes controlados e primeiros acessos.

Na passagem para ambientes mais estáveis ou compartilhados, a configuração costuma ser revista em pontos como banco de dados, persistência, autenticação, exposição externa, observabilidade e integração com serviços comuns da plataforma.
Essa revisão não altera a função principal do componente, mas ajusta a forma como ele se encaixa nas exigências operacionais do ambiente.

Por esse motivo, as páginas específicas dos componentes descrevem primeiro o comportamento inicial e depois os blocos de configuração que normalmente exigem atenção em laboratório, homologação ou produção.

Namespace

Cada componente pode ser instalado em um namespace dedicado ou compartilhar um namespace com outros componentes, de acordo com a estratégia do ambiente:

Terminal input

```
# Namespace dedicado (exemplo ilustrativo)
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <DEDICATED_NAMESPACE> --create-namespace

# Mesmo chart em namespace compartilhado com outros componentes
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <SHARED_NAMESPACE> --create-namespace
```

Recursos (CPU e memória)

Muitos charts permitem configurar `requests` e `limits` de CPU e memória para os principais pods.
Exemplo de estrutura:

© Copyright Tecnisys 2026. All rights reserved.

```
<CHART_VALUES_KEY>:
  resources:
    requests:
      cpu: "500m"
      memory: "1Gi"
    limits:
      cpu: "2"
      memory: "4Gi"
```

> **RECOMENDAÇÃO**
>
> Sempre defina `requests` e `limits` para garantir que o Kubernetes faça agendamento adequado dos pods e evite problemas de falta de recursos.

Persistência de dados

Os charts que requerem armazenamento persistente costumam expor opções de PVC (*Persistent Volume Claim*).
Exemplo de estrutura:

```
<CHART_VALUES_KEY>:
  persistence:
    enabled: true
    size: 10Gi
    storageClassName: "<STORAGE_CLASS>"
    accessMode: ReadWriteOnce
```

> **ATENÇÃO**
>
> Para componentes que executam em múltiplos nós e precisam de acesso concorrente ao mesmo volume, utilize `ReadWriteMany` (RWX) no `accessMode`. Certifique-se de que seu StorageClass suporta este modo.

Exposição HTTP externa

O TDP suporta dois mecanismos alternativos para expor componentes via HTTP/HTTPS:

© Copyright Tecnisys 2026. All rights reserved.

Mecanismo	Chave de ativação	Quando usar
Ingress	`TDP-Settings.gateway.ingress.enabled: true`	Clusters com Ingress Controller instalado (nginx, traefik, HAProxy...)
Gateway API	`TDP-Settings.gateway.gatewayApi.enabled: true`	Clusters com suporte a Gateway API v1 (Envoy Gateway, Istio...)

Ative apenas um mecanismo por vez. Após ativar, configure os parâmetros específicos do componente no chart correspondente (hostname, TLS, `ingressClassName` ou `gatewayClassName`).

Para pré-requisitos completos, exemplos por componente, TLS e validação, consulte Exposição externa: Ingress e Gateway API.

Serviços (ClusterIP, NodePort e LoadBalancer)

Um **Service** do Kubernetes cria um ponto de acesso estável para os Pods de um componente. Mesmo que os Pods sejam recriados ou mudem de IP, o Service mantém um endereço lógico para que outros componentes consigam se conectar.

Quando o chart expõe opções de Service, o campo `service.type` define como esse acesso será disponibilizado:

Tipo	Para que serve	Quando usar
`ClusterIP`	Expõe o serviço apenas dentro do cluster Kubernetes.	Padrão recomendado para comunicação interna entre componentes.
`NodePort`	Abre uma porta em cada nó do cluster, permitindo acesso externo pelo IP do nó.	Útil para testes ou ambientes sem Ingress/Gateway, mas

© Copyright Tecnisys 2026. All rights reserved.

Tipo	Para que serve	Quando usar
		menos indicado para produção.
`LoadBalancer`	Solicita um balanceador externo à infraestrutura do cluster.	Usado quando o ambiente Kubernetes oferece integração com load balancer externo.

```
<CHART_VALUES_KEY>:
  service:
    type: ClusterIP
```

Na maioria dos componentes TDP, `ClusterIP` é suficiente, porque a exposição externa deve ser tratada preferencialmente por Ingress ou Gateway API quando houver acesso HTTP/HTTPS pelo navegador ou por clientes externos.

Gerenciamento de secrets

Os charts do TDP seguem boas práticas de segurança para credenciais e dados sensíveis.

Kubernetes Secrets

Credenciais devem ser armazenadas em Kubernetes Secrets, nunca em texto plano nos arquivos de values:

Terminal input

```
kubectl -n <NAMESPACE> create secret generic meu-secret \
  --from-literal=password='<PASSWORD>'
```

Referência a Secrets no arquivo de valores

No arquivo de valores, referencie Secrets existentes em vez de colocar senhas em texto plano:

© Copyright Tecnisys 2026. All rights reserved.

```
passwordSecret:
  name: "meu-secret"
  key: "password"
```

> **IMPORTANTE**
>
> Nunca armazene credenciais em arquivos de valores que serão versionados no Git.
> Utilize Kubernetes Secrets, Sealed Secrets ou ferramentas como HashiCorp Vault.

TDP-Settings

Diversos charts do TDP utilizam a seção `TDP-Settings` para configuração de serviços compartilhados, como banco de dados externo e conexões S3:

```
TDP-Settings:
  externalDatabase:
    enabled: true
    recreate: false
    externalSecret:
      releaseName: "<POSTGRESQL_RELEASE>"
      area: "<SERVICE_KEY>"

  s3Connection:
    enabled: true
    secretName: "<S3_SECRET_NAME>"
    uri: "https://<S3_ENDPOINT>"
```

> **CAMPO `recreate`**
>
> O campo `recreate` controla se o banco de dados do componente deve ser recriado durante uma reinstalação:
>
> - `false` (padrão): mantém os dados existentes. Use em ambientes produtivos.
> - `true`: apaga e recria o banco. **Use quando houver certeza de que o banco de dados pode ser removido.**

© Copyright Tecnisys 2026. All rights reserved.

Decisões de ambiente que afetam a configuração

Antes de configurar os componentes TDP, identifique as características do ambiente que influenciam os valores dos charts, como armazenamento persistente, orquestrador, política de segurança, banco de dados compartilhado e serviços externos.

Registry de Helm Charts

Os charts TDP são distribuídos via registry OCI da Tecnisys.

Registry	Uso
`oci://registry.tecnisys.com.br/tdp/charts/`	Caminho OCI dos charts TDP estáveis e homologados (cada chart no subcaminho, por exemplo `.../charts/tdp-cloudbeaver`)

> **PRÉ-REQUISITO DE INSTALAÇÃO**
>
> A autenticação no registry é um pré-requisito de instalação, não uma configuração do componente.
> Para o procedimento completo de login e validação de acesso, consulte Pré-requisitos.

Decisões de armazenamento

Os componentes TDP que usam persistência (bancos de dados, logs, DAGs, cache) requerem que o cluster Kubernetes tenha pelo menos uma **StorageClass** disponível.

A StorageClass define **como** o cluster provisiona volumes persistentes (PVCs). Exemplos comuns:

© Copyright Tecnisys 2026. All rights reserved.

Tipo	Descrição	Quando usar
Local	Armazenamento direto no nó (ex.: `local-path-provisioner`)	Desenvolvimento, testes, ambientes não-críticos
NFS	Network File System compartilhado	Ambientes onde múltiplos nós precisam acessar o mesmo volume
Cloud	Managed storage (ex.: EBS na AWS, AKS, GKE)	Produção em cloud
SAN/Storage dedicado	Storage externo (ex.: NFS enterprise, Ceph)	Produção on-premises com SLA de disponibilidade

VERIFICAR STORAGECLASS DISPONÍVEL

No seu cluster, liste as StorageClasses e identifique qual usar:

Terminal input

```
kubectl get storageclass
```

Exemplo de saída:

```
NAME            PROVISIONER             RECLAIMPOLICY
VOLUMEBINDINGMODE
local-path      rancher.io/local-path   Delete
WaitForFirstConsumer
nfs-fast        nfs.provisioner.io      Delete          Immediate
```

Se nenhuma StorageClass estiver marcada com `*` (padrão), identifique qual usar e configure-a nos valores do chart:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-componente:
  persistence:
    storageClassName: "nfs-fast"  # Use o nome da StorageClass do seu
cluster
```

STORAGECLASS E READWRITEMANY (RWX)

Alguns componentes (ex.: Airflow com `KubernetesExecutor`, NiFi) executam múltiplas réplicas ou pods concorrentes que precisam **compartilhar** o mesmo volume.

Nesses casos, verifique se a StorageClass suporta `ReadWriteMany` (RWX):

Terminal input

```
kubectl describe storageclass <STORAGE_CLASS>
```

Se a StorageClass suporta apenas `ReadWriteOnce` (RWO), consulte a página de configuração do componente para saber se RWO é suficiente ou se você precisa de uma StorageClass diferente.

Compatibilidade do orquestrador e ajustes de segurança

Desde a release **3.0.1**, os componentes TDP suportam:

- **Red Hat OpenShift** 4.19+
- **Rancher Manager** 2.10.x+

Além do Kubernetes 1.32+.

OpenShift: Considerações gerais

O OpenShift é uma distribuição enterprise do Kubernetes que adiciona camadas de segurança acima do padrão Kubernetes.
Uma delas são as **Security Context Constraints (SCC)**, que restringem:

- Quais UIDs/GIDs os containers podem usar
- Se containers podem rodar como root

© Copyright Tecnisys 2026. All rights reserved.



- Quais capabilities do Linux são permitidas

Padrão: deixar UID/GID vazios

Para compatibilidade com o SCC `restricted` do OpenShift, deixe os campos `uid` e `gid` sem valor (`~` em YAML):

```
# No values.yaml, deixe:
uid: ~
gid: ~
```

Isso permite que o SCC do OpenShift **atribua automaticamente** um UID dentro do intervalo reservado do namespace, sem conflitos.

Padrão: `global.compatibility.openshift.adaptSecurityContext: force`

Vários charts TDP (e charts Bitnami empacotados) usam este parâmetro para se adaptar ao modelo de segurança do OpenShift:

```
tdp-componente:
  global:
    compatibility:
      openshift:
        adaptSecurityContext: force
```

Isso força o chart a remover restrições de segurança incompatíveis com o SCC restrito.

Role Binding automático para SCC

Se o componente precisar de privilégios acima do `restricted` padrão (ex.: root para instalar pacotes), o chart pode criar um RoleBinding que adiciona as ServiceAccounts ao SCC `anyuid`:

```
rbac:
  createSCCRoleBinding: true
```

© Copyright Tecnisys 2026. All rights reserved.



> **IMPORTANTE**
>
> Cada componente pode ter requisitos específicos de OpenShift.
> Consulte a página de configuração do componente (ex.: OpenMetadata) para ver se há seções dedicadas a "Instalação no OpenShift".

Rancher: nota sobre compatibilidade

O Rancher Manager é uma plataforma que gerencia múltiplos clusters Kubernetes (e OpenShift).
A compatibilidade com Rancher 2.10.x+ significa que os componentes TDP:

- Podem ser implantados em clusters gerenciados pelo Rancher
- Seguem as mesmas práticas de segurança e configuração que no Kubernetes padrão
- Herdam as políticas de rede e segurança do Rancher quando aplicáveis

Nenhuma configuração adicional específica do Rancher é necessária além das configurações usuais de Kubernetes.

PostgreSQL interno versus externo

Vários componentes TDP precisam de um banco relacional para armazenar seus próprios metadados.

O chart de cada componente suporta dois modos:

Modo	Descrição	Quando usar
PostgreSQL embutido (`postgres.enabled: true`)	Subchart instalado junto com o componente, autocontido	Desenvolvimento, testes isolados, ambientes temporários
PostgreSQL externo (`postgres.enabled: false`)	Usa o `tdp-postgresql` já instalado no cluster	Produção, ambientes compartilhados

> **POR QUE USAR POSTGRESQL EXTERNO EM PRODUÇÃO?**
>
> Com PostgreSQL externo e compartilhado (`tdp-postgresql`), centralizam-se:

© Copyright Tecnisys 2026. All rights reserved.



- **Backups**: um único ponto de backup para todos os bancos de metadados da plataforma
- **Monitoramento**: métricas e alertas em um único lugar
- **Dimensionamento**: ajusta `max_connections` e recursos uma única vez para toda a plataforma

O PostgreSQL embutido é conveniente mas isolado — cada componente gerencia seu próprio banco, o que aumenta o custo operacional em produção.

CLOUDBEAVER E METADADOS EM POSTGRESQL

O chart **CloudBeaver** pode guardar **metadados da aplicação** (configurações da UI, usuários locais, etc.) em um PostgreSQL **externo** além do volume de *workspace* (`database.enabled: true`), alinhado ao modo «PostgreSQL externo» desta seção.
Passos e exemplos: Configuração — CloudBeaver.

Armazenamento S3-compatível (Ozone, MinIO ou outro endpoint)

Alguns componentes TDP precisam acessar armazenamento de objetos S3-compatível para ler ou gravar arquivos (dados de entrada, outputs de pipelines, logs, DAGs).

No TDP Kubernetes, o backend S3 nativo da stack é o **Apache Ozone** (`tdp-ozone`), via seu S3 Gateway — mas qualquer endpoint compatível com S3 (MinIO ou um serviço de nuvem) pode ser usado no lugar; o mecanismo de conexão é o mesmo.

Quando um componente tem `TDP-Settings.s3Connection.enabled: true`, o chart cria automaticamente um Kubernetes Secret com os parâmetros de conexão (endpoint, chave de acesso, chave secreta).

Esse Secret é então referenciado pelos operadores e conectores do componente para acessar o endpoint S3 configurado — Ozone, MinIO ou outro.

QUANDO ESSA CONFIGURAÇÃO É NECESSÁRIA?

Configure a conexão S3 quando o componente precisar:

- Ler dados de entrada armazenados no S3 (ex.: Spark lendo Parquet)

© Copyright Tecnisys 2026. All rights reserved.

- Gravar outputs no S3 (ex.: resultados de pipelines Airflow)
- Manter arquivos entre execuções (ex.: DAGs do Airflow armazenados no S3)

Se o componente for usado apenas com dados em memória ou em PVCs locais, a conexão S3 não é necessária.

PRÓXIMO PASSO

Para acessar a configuração de um componente específico, volte ao índice de configuração.

© Copyright Tecnisys 2026. All rights reserved.

Exposição externa

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Objetivo da página

Esta página explica como expor interfaces HTTP/HTTPS de componentes do TDP Kubernetes para fora do cluster, usando **Ingress** ou **Gateway API**.

Os dois modelos resolvem o mesmo objetivo — publicar tráfego HTTP/HTTPS com hostname estável — mas usam recursos Kubernetes diferentes, dependem de controladores distintos e têm blocos de configuração separados nos charts TDP.

Quando expor um componente externamente

Exponha um componente quando ele precisa ser acessado por usuários, navegadores, ferramentas externas ou integrações HTTP/HTTPS fora do cluster Kubernetes — por exemplo, quando você quer um endereço estável como `https://superset.empresa.com` em vez de depender de comandos temporários ou portas locais.

Modo	Quando usar
Service interno (`<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local`)	Comunicação entre componentes dentro do cluster
`kubectl port-forward`	Acesso temporário para teste ou diagnóstico — depende do terminal, não é compartilhado
Ingress	Exposição estável com hostname HTTP/HTTPS usando um Ingress Controller
Gateway API	Exposição estável com hostname HTTP/HTTPS usando um Gateway Con-

© Copyright Tecnisys 2026. All rights reserved.

Modo	Quando usar
	troller e recurso `Gateway`

Ingress vs Gateway API

Ingress e Gateway API são alternativas para publicar tráfego HTTP/HTTPS para fora do cluster. A escolha do modo é feita no bloco `TDP-Settings.gateway`, mas a configuração detalhada **não é igual** nos dois modelos.

Aspecto	Ingress	Gateway API
Recurso Kubernetes criado	`Ingress`	`HTTPRoute` (e opcionalmente `Gateway`)
Controlador necessário	Ingress Controller (NGINX, Traefik...)	Gateway Controller (Envoy Gateway, Istio...)
Recurso de entrada no cluster	Gerenciado pelo Ingress Controller	Recurso `Gateway` (criado pelo chart ou pré-existente)
Associação ao controlador	`ingressClassName`	`gatewayApi.gatewayClassName` e `gatewayApi.parentRefs`
Hostname	`hosts[].host` ou `hostname`	`hostnames[]`
Caminhos	`paths[]`	`rules[].matches[]`
Backend	Definido pelo template do chart ou pelos paths	`backendRefs[]`

© Copyright Tecnisys 2026. All rights reserved.

Aspecto	Ingress	Gateway API
TLS	`ingress.tls` ou bloco equivalente do componente	No `Gateway` ou em `certificateRefs`

Regra de exclusividade

> ⚠ **USE APENAS UM MODO POR COMPONENTE**
>
> `TDP-Settings.gateway.ingress.enabled` e `TDP-Settings.gateway.gatewayApi.enabled` são mutuamente exclusivos.
>
> - **Ingress:** `TDP-Settings.gateway.ingress.enabled: true` e `gatewayApi.enabled: false`
> - **Gateway API:** `TDP-Settings.gateway.gatewayApi.enabled: true` e `ingress.enabled: false`
>
> Nunca habilite os dois ao mesmo tempo para o mesmo componente.

Pré-requisitos

Comuns a ambos os modos:

- Namespace do componente criado e chart instalado (ou arquivo de valores preparado).
- DNS ou `/etc/hosts` apontando o hostname para o endereço do controlador de entrada.

Ingress:

- Ingress Controller instalado no cluster (NGINX, Traefik ou equivalente).
- `ingressClassName` conhecido (`kubectl get ingressclass`).

Gateway API:

- Gateway Controller instalado no cluster.
- `GatewayClass` disponível (`kubectl get gatewayclass`).

© Copyright Tecnisys 2026. All rights reserved.

- CRD `gateways.gateway.networking.k8s.io` instalada.

Para testes locais, adicione o hostname ao `/etc/hosts`:

```
# /etc/hosts
<CONTROLLER_IP>   <COMPONENT_HOSTNAME>
```

Padrão comum de configuração

A chave de controle é `TDP-Settings.gateway`. A estrutura detalhada varia por chart — consulte a tabela abaixo, a página de configuração do componente ou o `values.yaml`:

Terminal input

```
helm show values oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> >
values-padrao.yaml
```

Exemplo:

Terminal input

```
helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver >
values-cloudbeaver.yaml
```

Ingress

Como funciona

No modo Ingress, o chart cria um recurso `Ingress` no cluster. Um **Ingress Controller** recebe o tráfego externo e o encaminha para o Service do componente com base no hostname e no path configurados.

Aplicar os valores

Adicione a configuração desejada ao seu arquivo de values e aplique via Helm:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace --timeout 10m \
  -f <VALUES_FILE>
```

Substitua `<CHART_NAME>` pelo nome do chart do componente (ex.: `tdp-jupyter`, `tdp-airflow`). O caminho OCI completo está na tabela de suporte acima ou na página de configuração do componente.

Teste local sem DNS

Se o hostname ainda não tiver entrada DNS, obtenha o IP do Ingress Controller e adicione ao `/etc/hosts`:

Terminal input

```
# Exemplo para ingress-nginx — ajuste namespace e nome do Service se diferente
kubectl get svc -n ingress-nginx ingress-nginx-controller \
  -o jsonpath='{.status.loadBalancer.ingress[0].ip}'

# Adicione ao /etc/hosts (substitua pelos valores reais)
echo "<CONTROLLER_IP> <COMPONENT_HOSTNAME>" | sudo tee -a /etc/hosts
```

Acesse em seguida: `http://<COMPONENT_HOSTNAME>`

Padrão comum (Jupyter, Kafka UI, ClickHouse, Airflow)

Componentes com `ingress.*` top-level seguem este modelo:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

ingress:
```

© Copyright Tecnisys 2026. All rights reserved.

```
  ingressClassName: <INGRESS_CLASS>
  hosts:
    - host: <COMPONENT_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []
```

JupyterHub

Para sessões longas de notebook, adicione as anotações de timeout ao Ingress:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

ingress:
  ingressClassName: <INGRESS_CLASS>
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
  hosts:
    - host: <JUPYTER_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []
```

Sem as anotações de timeout, células com execução demorada podem resultar em timeout do gateway (504).

Airflow

Além do Ingress, configure proxy e CORS no subchart:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
    gatewayApi:
      enabled: false

ingress:
  ingressClassName: <INGRESS_CLASS>
  annotations: {}
  hosts:
    - host: <AIRFLOW_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []

tdp-airflow:
  config:
    webserver:
      base_url: "http://<AIRFLOW_HOSTNAME>"
      x_forwarded_proto: "http"
      x_forwarded_host: "<AIRFLOW_HOSTNAME>"
      x_forwarded_port: "<SERVICE_PORT>"
    api:
      cors_enabled: "True"
      cors_allow_origin: "*"
```

Argo CD

Usa estrutura própria em `tdp-argo.server.ingress` — **não** o bloco `ingress` top-level:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-argo:
  configs:
    params:
      server.insecure: "true"
    cm:
      url: http://<ARGOCD_HOSTNAME>
  server:
    ingress:
```

© Copyright Tecnisys 2026. All rights reserved.

```
      enabled: true
      controller: generic
      hostname: <ARGOCD_HOSTNAME>
      ingressClassName: <INGRESS_CLASS>
      annotations: {}
      path: /
      pathType: Prefix
```

OpenShift

No OpenShift, use `ingressClassName: openshift-default` e o hostname segue o padrão de rotas do cluster:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-argo:
  configs:
    params:
      server.insecure: "true"
    cm:
      url: http://argocd-<NAMESPACE>.apps.<CLUSTER_DOMAIN>
  server:
    ingress:
      enabled: true
      controller: generic
      ingressClassName: openshift-default
      hostname: argocd-<NAMESPACE>.apps.<CLUSTER_DOMAIN>
      path: /
      pathType: Prefix
      annotations: {}
```

ClickHouse

Ingress expõe a interface **HTTP** (8123). Configure `extraUsers` para permitir conexões da rede do Ingress:

© Copyright Tecnisys 2026. All rights reserved.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-clickhouse:
  clickhouse:
    extraUsers: |
      <yandex>
        <profiles>
          <default>

<query_cache_system_table_handling>ignore</query_cache_system_table_handling>
          </default>
        </profiles>
        <users>
          <default>
            <networks>
              <ip>::/0</ip>
            </networks>
          </default>
        </users>
      </yandex>

ingress:
  ingressClassName: <INGRESS_CLASS>
  annotations: {}
  hosts:
    - host: <CLICKHOUSE_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []
```

> **TCP NATIVO (PORTA 9000)**
>
> A porta TCP nativa do ClickHouse **não** pode ser exposta via Ingress. Use `port-forward`, `LoadBalancer` ou `NodePort` para acesso TCP.

© Copyright Tecnisys 2026. All rights reserved.



Após a exposição, acesse as interfaces web do ClickHouse:

- Play UI: `http://<CLICKHOUSE_HOSTNAME>/play`
- Dashboard: `http://<CLICKHOUSE_HOSTNAME>/dashboard`

CloudBeaver

Use `tdp-cloudbeaver.ingress` para publicar a interface web do CloudBeaver:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-cloudbeaver:
  ingress:
    ingressClassName: <INGRESS_CLASS>
    annotations:
      nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
      nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    hosts:
      - host: <CLOUDBEAVER_HOSTNAME>
        paths:
          - path: /
            pathType: Prefix
    tls: false
```

Exemplos de hostname:

- Kubernetes padrão: `cloudbeaver.tdp.local`
- OpenShift: `cloudbeaver-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Kafka UI

Expõe apenas a **Kafka UI** — não os brokers Kafka:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
    gatewayApi:
      enabled: false

ingress:
  ingressClassName: <INGRESS_CLASS>   # ex.: nginx, openshift-default
  annotations: {}
  hosts:
    - host: <KAFKA_UI_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `kafka-ui.tdp.local`
- OpenShift: `kafka-ui-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

> **KAFKA UI ≠ BROKERS KAFKA**
>
> O Ingress publica a interface web da Kafka UI. Produtores e consumidores externos continuam precisando de configuração adicional para acessar os brokers (listeners Kafka), que não é coberta por Ingress/Gateway API neste chart.

NiFi

O NiFi usa `ingress.rules` (não `ingress.hosts`) como estrutura de Ingress, e requer anotações de afinidade de sessão.

Kubernetes padrão (HTTP)

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

ingress:
  enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
  ingressClassName: <INGRESS_CLASS>
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/affinity-mode: "persistent"
    nginx.ingress.kubernetes.io/session-cookie-name: "INGRESSCOOKIE"
    nginx.ingress.kubernetes.io/session-cookie-path: "/"
  rules:
    - host: <NIFI_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: tdp-nifi-service
              port:
                number: 8080
  tls: []
```

Kubernetes padrão (HTTPS)

Quando `nifiCluster.security.protocol: "https"`, altere a anotação e a porta:

```
ingress:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
  rules:
    - host: <NIFI_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: tdp-nifi-service
              port:
                number: 8443
```

OpenShift

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
    gatewayApi:
      enabled: false

ingress:
  enabled: true
  ingressClassName: openshift-default
  annotations:
    haproxy.router.openshift.io/balance: "cookie"
    haproxy.router.openshift.io/cookie_name: "INGRESSCOOKIE"
  rules:
    - host: nifi-<NAMESPACE>.apps.<CLUSTER_DOMAIN>
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: tdp-nifi-service
              port:
                number: 8080
  tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `nifi.tdp.local`
- OpenShift: `nifi-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

> **NOME DO ARQUIVO DE VALORES**
>
> Use o nome que preferir para o arquivo de valores de Ingress e passe com `-f <VALUES_FILE>`.

OpenMetadata

No OpenMetadata, a configuração de Ingress fica sob **`tdp-openmetadata.ingress.*`** (dentro do subchart), não no `ingress.*` raiz.

Kubernetes padrão

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
    gatewayApi:
      enabled: false

tdp-openmetadata:
  ingress:
    className: <INGRESS_CLASS>
    annotations: {}
    hosts:
      - host: <OPENMETADATA_HOSTNAME>
        paths:
          - path: /
            pathType: Prefix
    tls: []
```

OpenShift

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-openmetadata:
  ingress:
    className: openshift-default
    annotations: {}
    hosts:
      - host: openmetadata-<NAMESPACE>.apps.<CLUSTER_DOMAIN>
        paths:
          - path: /
            pathType: Prefix
    tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `openmetadata.tdp.local`
- OpenShift: `openmetadata-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Ozone

O Ozone expõe quatro endpoints independentes: S3 Gateway REST, S3 Gateway Web UI, Ozone Manager UI e SCM UI. Cada endpoint tem seu próprio bloco de Ingress:

© Copyright Tecnisys 2026. All rights reserved.



```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-ozone:
  ingress:
    s3g:
      rest:
        enabled: true
        ingressClassName: <INGRESS_CLASS>
        annotations: {}
        hosts:
          - host: <OZONE_S3_REST_HOSTNAME>
            paths:
              - path: /
                pathType: Prefix
        tls: []
      web:
        enabled: true
        ingressClassName: <INGRESS_CLASS>
        annotations: {}
        hosts:
          - host: <OZONE_S3_WEB_HOSTNAME>
            paths:
              - path: /
                pathType: Prefix
        tls: []
    om:
      enabled: true
      ingressClassName: <INGRESS_CLASS>
      annotations: {}
      hosts:
        - host: <OZONE_OM_HOSTNAME>
          paths:
            - path: /
              pathType: Prefix
      tls: []
    scm:
      enabled: true
      ingressClassName: <INGRESS_CLASS>
      annotations: {}
```

© Copyright Tecnisys 2026. All rights reserved.

```
      hosts:
        - host: <OZONE_SCM_HOSTNAME>
          paths:
            - path: /
              pathType: Prefix
      tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `ozone-s3.tdp.local`, `ozone-s3-ui.tdp.local`, `ozone-om.tdp.local`, `ozone-scm.tdp.local`
- OpenShift: `ozone-s3-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-s3-ui-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-om-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-scm-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Ranger

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

ingress:
  ingressClassName: <INGRESS_CLASS>
  annotations: {}
  hosts:
    - host: <RANGER_HOSTNAME>
      paths:
        - path: /
          pathType: Prefix
  tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `ranger.tdp.local`
- OpenShift: `ranger-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Superset

© Copyright Tecnisys 2026. All rights reserved.



No Superset, a configuração de Ingress fica sob **`tdp-superset.ingress.*`** (dentro do subchart), não no `ingress.*` raiz.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

tdp-superset:
  ingress:
    enabled: true
    ingressClassName: <INGRESS_CLASS>
    annotations: {}
    path: /
    pathType: Prefix
    hosts:
      - <SUPERSET_HOSTNAME>
    tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `superset.tdp.local`
- OpenShift: `superset-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Trino

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

ingress:
  enabled: true
  ingressClassName: <INGRESS_CLASS>
  annotations: {}
  path: /
  pathType: Prefix
  hosts:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    - <TRINO_HOSTNAME>
  tls: []
```

Exemplos de hostname:

- Kubernetes padrão: `trino.tdp.local`
- OpenShift: `trino-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Spark

O Spark requer parâmetros adicionais de reverse proxy para que a UI funcione corretamente via Ingress. A configuração fica em `spark.ingress.*` (não no `ingress.*` raiz).

```
TDP-Settings:
  gateway:
    ingress:
      enabled: true
    gatewayApi:
      enabled: false

spark:
  ingress:
    enabled: true
    ingressClassName: <INGRESS_CLASS>
    hostname: <SPARK_HOSTNAME>
    path: /
    pathType: Prefix
  master:
    configOptions: "-Dspark.ui.reverseProxy=true -
Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
  worker:
    configOptions: "-Dspark.ui.reverseProxy=true -
Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
  sparkConf:
    "spark.ui.reverseProxy": "true"
    "spark.ui.reverseProxyUrl": "http://<SPARK_HOSTNAME>"
```

Exemplos:

- Kubernetes padrão: `<INGRESS_CLASS>: nginx`, `<SPARK_HOSTNAME>: spark.tdp.local`

© Copyright Tecnisys 2026. All rights reserved.



- OpenShift: `<INGRESS_CLASS>: openshift-default`, `<SPARK_HOSTNAME>: spark-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

TLS com Ingress

Nos exemplos anteriores, `tls: []` indica que nenhuma entrada TLS foi definida. Para habilitar HTTPS no Ingress, substitua o array vazio por uma lista no formato de `spec.tls` do Kubernetes:

```
ingress:
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
  tls:
    - secretName: <SECRET_NAME>
      hosts:
        - <COMPONENT_HOSTNAME>
```

Use o mesmo formato no bloco equivalente do componente quando o chart não usa `ingress.*` top-level, por exemplo `tdp-ozone.ingress.s3g.rest.tls` ou `tdp-openmetadata.ingress.tls`.

Se o Secret TLS for criado manualmente, ele deve existir no namespace do componente antes da instalação ou atualização. Com cert-manager, a anotação do Ingress solicita a emissão/renovação automática e o controlador grava o Secret indicado em `secretName`.

Validação

Terminal input

```
kubectl get ingress -n <NAMESPACE>
kubectl describe ingress <INGRESS_NAME> -n <NAMESPACE>
curl -I http://<COMPONENT_HOSTNAME>
```

Resolução de problemas

© Copyright Tecnisys 2026. All rights reserved.

Problema	Diagnóstico	Solução
Ingress não criado	`kubectl get ingress -n <NAMESPACE>` não lista	Verifique `TDP-Settings.gateway.ingress.enabled: true` e reaplique o Helm
`ingressClassName` incorreto	`kubectl describe ingress` mostra eventos de erro	Execute `kubectl get ingressclass`
Hostname não resolve	`curl` falha com DNS loo-kup error	Configure `/etc/hosts` ou DNS para o IP do Ingress Controller
404 Not Found	Ingress criado mas re-torna 404	Verifique `path` e `pathType`
502 Bad Gateway	Ingress criado mas ser-viço não responde	Execute `kubectl get endpoints -n <NAMESPACE>`
ClickHouse `AUTHENTICATION_FAILED`	Acesso via Ingress rejeitado	Configure `tdp-clickhouse.clickhouse.extraUsers` com `<ip>::/0</ip>`
Airflow providers não carregam	UI abre mas recursos falham	Verifique `tdp-airflow.config.webserver.base_url` e headers `x_forwarded_*`
Argo CD redirect loop	UI não carrega corretamente	Verifique `server.insecure: "true"` e `configs.cm.url`

Gateway API

Como funciona

© Copyright Tecnisys 2026. All rights reserved.

No modo Gateway API, o chart cria um recurso `HTTPRoute` e, quando `gatewayApi.gateway.enabled: true`, também um recurso `Gateway`. Um **Gateway Controller** processa esses recursos e encaminha o tráfego para o Service do componente.

Aplicar os valores

Adicione a configuração desejada ao seu arquivo de values e aplique via Helm:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/<CHART_NAME> \
  -n <NAMESPACE> --create-namespace --timeout 10m \
  -f <VALUES_FILE>
```

Substitua `<CHART_NAME>` pelo nome do chart do componente (ex.: `tdp-jupyter`, `tdp-airflow`).

Acesso via Gateway API

Após o deploy, obtenha o IP do Gateway Controller e adicione ao `/etc/hosts` para teste local:

Terminal input

```
# Verifique o nome e namespace do Gateway criado
kubectl get gateway -A

# Obtenha o IP do LoadBalancer do Gateway Controller (ajuste namespace/nome)
kubectl get svc -n <GATEWAY_NAMESPACE> <GATEWAY_SERVICE> \
  -o jsonpath='{.status.loadBalancer.ingress[0].ip}'

# Adicione ao /etc/hosts (substitua pelos valores reais)
echo "<CONTROLLER_IP> <COMPONENT_HOSTNAME>" | sudo tee -a /etc/hosts
```

Acesse em seguida: `http://<COMPONENT_HOSTNAME>`

Padrão comum de Gateway API

© Copyright Tecnisys 2026. All rights reserved.



Quando o chart cria o `Gateway` automaticamente, a estrutura base é:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <COMPONENT_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  <APPLICATION_NAME>:
    enabled: true
    hostnames:
      - <COMPONENT_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - port: <SERVICE_PORT>
    tls:
      certificateRefs: []
```

Substitua `<APPLICATION_NAME>` pela chave do chart (`airflow`, `server`, `clickhouse`, `cloudbeaver`, `jupyter`, `kafkaUi`).

Airflow

```
TDP-Settings:
  gateway:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <AIRFLOW_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  airflow:
    enabled: true
    hostnames:
      - <AIRFLOW_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8080
    tls:
      certificateRefs: []

tdp-airflow:
  config:
    webserver:
      base_url: "http://<AIRFLOW_HOSTNAME>"
      x_forwarded_proto: "http"
      x_forwarded_host: "<AIRFLOW_HOSTNAME>"
      x_forwarded_port: "<SERVICE_PORT>"
    api:
      cors_enabled: "True"
      cors_allow_origin: "*"
```

Argo CD

© Copyright Tecnisys 2026. All rights reserved.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

tdp-argo:
  configs:
    params:
      server.insecure: "true"
    cm:
      url: http://<ARGOCD_HOSTNAME>

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <ARGOCD_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
      namespace: <GATEWAY_NAMESPACE>
  server:
    enabled: true
    hostnames:
      - <ARGOCD_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 80
    tls:
      certificateRefs: []
```

ClickHouse

© Copyright Tecnisys 2026. All rights reserved.

Requer `extraUsers` (mesmo requisito do Ingress):

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

tdp-clickhouse:
  clickhouse:
    extraUsers: |
      <yandex>
        <profiles>
          <default>

<query_cache_system_table_handling>ignore</query_cache_system_table_handling>
          </default>
        </profiles>
        <users>
          <default>
            <networks>
              <ip>::/0</ip>
            </networks>
          </default>
        </users>
      </yandex>

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <CLICKHOUSE_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  clickhouse:
    enabled: true
    hostnames:
      - <CLICKHOUSE_HOSTNAME>
```

© Copyright Tecnisys 2026. All rights reserved.

```
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8123
    tls:
      certificateRefs: []
```

Após a exposição, acesse as interfaces web do ClickHouse:

- Play UI: `http://<CLICKHOUSE_HOSTNAME>/play`
- Dashboard: `http://<CLICKHOUSE_HOSTNAME>/dashboard`

CloudBeaver

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <CLOUDBEAVER_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  cloudbeaver:
    enabled: true
    hostnames:
      - <CLOUDBEAVER_HOSTNAME>
    rules:
      - matches:
```

© Copyright Tecnisys 2026. All rights reserved.

```
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8978
    tls:
      certificateRefs: []
```

Jupyter (padrão comum)

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <JUPYTER_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  jupyter:
    enabled: true
    hostnames:
      - <JUPYTER_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 80
```

© Copyright Tecnisys 2026. All rights reserved.

```
    tls:
      certificateRefs: []
```

Kafka UI

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <KAFKA_UI_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  kafkaUi:
    enabled: true
    hostnames:
      - <KAFKA_UI_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 80
    tls:
      certificateRefs: []
```

> **KAFKA UI ≠ BROKERS KAFKA**

© Copyright Tecnisys 2026. All rights reserved.



> O Gateway API publica apenas a Kafka UI. Os brokers Kafka permanecem acessíveis internamente via listeners configurados no chart — não via HTTPRoute.

OpenMetadata

O backend do OpenMetadata opera na porta 8585 (API e interface web).

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
        hostname: <OPENMETADATA_HOSTNAME>
  parentRefs:
    - name: <GATEWAY_NAME>
  openmetadata:
    enabled: true
    hostnames:
      - <OPENMETADATA_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8585
    tls:
      certificateRefs: []
```

© Copyright Tecnisys 2026. All rights reserved.



Exemplos de hostname:

- Kubernetes padrão: `openmetadata.tdp.local`
- OpenShift: `openmetadata-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

NiFi

O backend do NiFi usa a porta 8080 (HTTP) ou 8443 (HTTPS), conforme `nifiCluster.security.protocol`.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  parentRefs:
    - name: <GATEWAY_NAME>
      namespace: <GATEWAY_NAMESPACE>
  nifi:
    enabled: true
    hostnames:
      - <NIFI_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8080
    tls:
      certificateRefs: []
```

> **NOME DO ARQUIVO DE VALORES**
>
> Use o nome que preferir para o arquivo de valores do Gateway API e passe com `-f <VALUES_FILE>`.

© Copyright Tecnisys 2026. All rights reserved.

Ozone

O Ozone expõe quatro endpoints via Gateway API: S3 REST, S3 Web UI, OM UI e SCM UI:

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

tdp-ozone:
  gatewayApi:
    parentRefs:
      - name: <GATEWAY_NAME>
        namespace: <GATEWAY_NAMESPACE>
    s3g:
      rest:
        enabled: true
        hostnames:
          - <OZONE_S3_REST_HOSTNAME>
        rules:
          - matches:
              - path:
                  type: PathPrefix
                  value: /
            backendRefs:
              - name: ""
                port: 9878
        tls:
          certificateRefs: []
      web:
        enabled: true
        hostnames:
          - <OZONE_S3_WEB_HOSTNAME>
        rules:
          - matches:
              - path:
                  type: PathPrefix
                  value: /
            backendRefs:
              - name: ""
                port: 19878
        tls:
```

© Copyright Tecnisys 2026. All rights reserved.

```
        certificateRefs: []
    om:
      enabled: true
      hostnames:
        - <OZONE_OM_HOSTNAME>
      rules:
        - matches:
            - path:
                type: PathPrefix
                value: /
          backendRefs:
            - name: ""
              port: 9874
      tls:
        certificateRefs: []
    scm:
      enabled: true
      hostnames:
        - <OZONE_SCM_HOSTNAME>
      rules:
        - matches:
            - path:
                type: PathPrefix
                value: /
          backendRefs:
            - name: ""
              port: 9876
      tls:
        certificateRefs: []
```

Exemplos de hostname:

- Kubernetes padrão: `ozone-s3.tdp.local`, `ozone-s3-ui.tdp.local`, `ozone-om.tdp.local`, `ozone-scm.tdp.local`
- OpenShift: `ozone-s3-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-s3-ui-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-om-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`, `ozone-scm-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Ranger

```
TDP-Settings:
  gateway:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
  parentRefs:
    - name: <GATEWAY_NAME>
  ranger:
    enabled: true
    hostnames:
      - <RANGER_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 6080
    tls:
      certificateRefs: []
```

Exemplos de hostname:

- Kubernetes padrão: `ranger.tdp.local`
- OpenShift: `ranger-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Superset

O Superset reutiliza um Gateway externo via `parentRefs` — não cria um `Gateway` próprio. Não inclui `gatewayClassName` nem `gateway.enabled`.

```
TDP-Settings:
  gateway:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  parentRefs:
    - name: <GATEWAY_NAME>
      namespace: <GATEWAY_NAMESPACE>
  superset:
    enabled: true
    hostnames:
      - <SUPERSET_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8088
    tls:
      certificateRefs: []
```

Exemplos de hostname:

- Kubernetes padrão: `superset.tdp.local`
- OpenShift: `superset-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Trino

O Trino reutiliza um Gateway externo via `parentRefs` — não cria um `Gateway` próprio. Não inclui `gatewayClassName` nem `gateway.enabled`.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  parentRefs:
```

© Copyright Tecnisys 2026. All rights reserved.



```
    - name: <GATEWAY_NAME>
      namespace: <GATEWAY_NAMESPACE>
  trino:
    enabled: true
    hostnames:
      - <TRINO_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 8080
    tls:
      certificateRefs: []
```

Exemplos de hostname:

- Kubernetes padrão: `trino.tdp.local`
- OpenShift: `trino-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Spark

Assim como no Ingress, o Spark requer parâmetros de reverse proxy também no modo Gateway API.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: true

gatewayApi:
  gatewayClassName: <GATEWAY_CLASS>
  gateway:
    enabled: true
    name: <GATEWAY_NAME>
    listeners:
      - name: web
        port: 8000
        protocol: HTTP
```

© Copyright Tecnisys 2026. All rights reserved.

```
  parentRefs:
    - name: <GATEWAY_NAME>
      namespace: <GATEWAY_NAMESPACE>
  spark:
    enabled: true
    hostnames:
      - <SPARK_HOSTNAME>
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /
        backendRefs:
          - name: ""
            port: 80
    tls:
      certificateRefs: []

spark:
  master:
    configOptions: "-Dspark.ui.reverseProxy=true -
Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
  worker:
    configOptions: "-Dspark.ui.reverseProxy=true -
Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
  sparkConf:
    "spark.ui.reverseProxy": "true"
    "spark.ui.reverseProxyUrl": "http://<SPARK_HOSTNAME>"
```

Exemplos de hostname:

- Kubernetes padrão: `spark.tdp.local`
- OpenShift: `spark-<NAMESPACE>.apps.<CLUSTER_DOMAIN>`

Referência rápida: chaves e portas

A chave `gatewayApi.<APPLICATION_NAME>` varia por chart — as sub-chaves e portas de backend são:

© Copyright Tecnisys 2026. All rights reserved.

Chart	Chave `gatewayApi`	Porta backend
tdp-airflow	`gatewayApi.airflow`	8080
tdp-argo	`gatewayApi.server`	80
tdp-clickhouse	`gatewayApi.clickhouse`	8123
tdp-cloudbeaver	`gatewayApi.cloudbeaver`	8978
tdp-jupyter	`gatewayApi.jupyter`	80
tdp-kafka	`gatewayApi.kafkaUi`	80
tdp-nifi	`gatewayApi.nifi`	8080 (HTTP) / 8443 (HTTPS)
tdp-openmetadata	`gatewayApi.openmetadata`	8585
tdp-ozone (S3 REST)	`gatewayApi.s3g.rest`	9878
tdp-ozone (S3 Web UI)	`gatewayApi.s3g.web`	19878
tdp-ozone (OM UI)	`gatewayApi.om`	9874
tdp-ozone (SCM UI)	`gatewayApi.scm`	9876
tdp-ranger	`gatewayApi.ranger`	6080
tdp-spark	`gatewayApi.spark`	80
tdp-superset	`gatewayApi.superset`	8088
tdp-trino	`gatewayApi.trino`	8080

TLS com Gateway API

© Copyright Tecnisys 2026. All rights reserved.

Terminação no Gateway (mais comum): o listener HTTPS do `Gateway` faz a terminação TLS — nenhuma configuração adicional é necessária no `HTTPRoute`.

`certificateRefs` por componente (quando suportado): TLS no bloco `gatewayApi.<APPLICATION_NAME>.tls.certificateRefs`.

Validação

Terminal input

```
kubectl get gatewayclass
kubectl get gateway -A
kubectl get httproute -n <NAMESPACE>
kubectl describe httproute <HTTPROUTE_NAME> -n <NAMESPACE>
curl -I http://<COMPONENT_HOSTNAME>
```

Resolução de problemas

Problema	Diagnóstico	Solução
`HTTPRoute` não criado	`kubectl get httproute -n <NAMESPACE>` não lista	Verifique `TDP-Settings.gateway.gatewayApi.enabled: true`
`Gateway` não criado	`kubectl get gateway -A` não lista o recurso	Verifique `gatewayApi.gateway.enabled: true`
`HTTPRoute` sem tráfego	Status inválido no describe	Confirme `parentRefs.name` apontando para o Gateway correto
Hostname não resolve	`curl` falha com DNS lookup error	DNS deve apontar para o IP do Gateway
`backendRefs.port` incorreto	Tráfego não chega ao componente	Confirme a porta do Service com `kubectl get svc -n <NAMESPACE>`

© Copyright Tecnisys 2026. All rights reserved.

Problema	Diagnóstico	Solução
ClickHouse `AUTHENTICATION_FAILED`	Acesso via Gateway rejeitado	Configure `extraUsers` com `<ip>::/0</ip>`
`gatewayclass` não encontrada	`kubectl get gatewayclass` retorna vazio	Instale o Gateway Controller no cluster

© Copyright Tecnisys 2026. All rights reserved.

Segurança — Padrões Gerais

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Esta página apresenta os principais padrões de segurança usados nos componentes TDP Kubernetes.

Ela não substitui as páginas específicas de segurança de cada componente. Use esta página para entender os conceitos gerais e consulte a página do componente para aplicar os parâmetros corretos, nomes de Secrets, arquivos de valores e exemplos completos.

Como usar esta página

Use esta tabela como mapa rápido de onde cada decisão de segurança deve ser tratada. Ela não é uma taxonomia rígida: a mesma configuração pode envolver mais de uma camada.

Dúvida ou decisão	O que verificar	Onde aprofundar
Credenciais sensíveis	Senhas, tokens, chaves de acesso, senhas de bind, keystores, truststores e referência a Secrets existentes	Credenciais e Kubernetes Secrets; página de segurança ou integração do componente
Login de usuários	LDAP, Active Directory, SSO, conta interna e escopo da autenticação	Autenticação de usuários; segurança do componente
TLS de entrada	HTTPS via Ingress ou Gateway API, hostname, certificado e Secret TLS	TLS no Ingress ou Gateway API; Configuração de Ingress

© Copyright Tecnisys 2026. All rights reserved.

Dúvida ou decisão	O que verificar	Onde aprofundar
TLS interno ou de integração	LDAPS, conexão segura com banco, S3, listeners, APIs internas, keystores e truststores	TLS na aplicação; TLS em integrações; segurança ou integração do componente
Permissões e políticas	Autorização da aplicação, políticas Ranger, usuários, grupos e aplicação das políticas em runtime	Autorização e políticas; Segurança - Ranger quando aplicável
Controle de rede	Origens e destinos permitidos, tráfego entre namespaces e exposição de Services internos	NetworkPolicy e controle de acesso à rede; página do componente
Auditoria	Logs de acesso, falhas de autenticação, operações sensíveis e rastreabilidade operacional	Auditoria e rastreabilidade; página do componente

Princípios gerais

Antes de configurar segurança em qualquer componente, observe estes princípios:

- não versionar senhas em texto plano em arquivos de valores;
- preferir Kubernetes Secrets para credenciais sensíveis;
- usar hostnames, certificados e Secrets compatíveis com o ambiente real;
- validar se a configuração pertence ao componente, ao Ingress ou a uma integração externa;
- consultar a página específica do componente antes de aplicar parâmetros de segurança.

Credenciais e Kubernetes Secrets

Credenciais sensíveis, como senhas de banco de dados, tokens, chaves de acesso, senhas de bind LDAP e senhas de keystore, devem ser armazenadas em Kubernetes

© Copyright Tecnisys 2026. All rights reserved.

Secrets.

Os componentes TDP geralmente permitem referenciar Secrets existentes por parâmetros como `existingSecret`, `secretKeyRef`, `passwordSecret.name` ou estruturas equivalentes documentadas no componente.

Prefira essa abordagem a fornecer senhas diretamente nos valores do componente.

Criar um Secret

Terminal input

```
kubectl -n <NAMESPACE> create secret generic <SECRET_NAME> --from-literal=
<CONFIG_KEY>='<CONFIG_VALUE>'
```

> **NUNCA VERSIONE SENHAS**
>
> Não inclua senhas em texto plano em arquivos de valores commitados no Git. Use referências a Secrets ou ferramentas próprias de gestão de Secrets.

Gestão avançada de Secrets

Em ambientes de produção, considere ferramentas especializadas para gestão de Secrets:

Ferramenta	Uso
Sealed Secrets	Permite versionar Secrets criptografados no Git
External Secrets Operator	Sincroniza Secrets a partir de cofres externos
HashiCorp Vault	Centraliza credenciais e políticas de acesso

Rotação e revogação de credenciais

© Copyright Tecnisys 2026. All rights reserved.

Credenciais de longa duração precisam de ciclo de rotação definido. Ao rotacionar:

- Atualize ou recrie o Secret Kubernetes com as novas credenciais.
- Reinicie os pods que consomem esse Secret (variáveis de ambiente ou arquivos montados).
- Valide que as aplicações estão operando com as novas credenciais antes de descartar as antigas.

A revogação prática em Kubernetes é operacional: remover ou substituir o Secret, reiniciar workloads e bloquear caminhos de rede quando necessário. O tempo de exposição após comprometimento depende de quão rapidamente os Secrets são atualizados e os pods são reiniciados.

Consulte a página de segurança do componente para saber quais Secrets e pods precisam ser reiniciados ao rotacionar credenciais específicas.

Autenticação de usuários

Alguns componentes TDP permitem autenticação integrada a LDAP, Active Directory, SSO ou outro provedor externo.

A forma de configuração varia conforme o componente. Ela pode envolver:

- parâmetros diretos nos valores do componente;
- arquivo adicional de valores aplicado como overlay Helm;
- configuração própria da aplicação;
- integração com provedor intermediário de autenticação;
- Kubernetes Secrets para armazenar credenciais de bind ou client secrets.

Por isso, não existe um único modelo LDAP ou SSO aplicável a todos os componentes.

> **LDAP/AD: DESATIVAR CONTA NÃO REVOGA CREDENCIAL DISTRIBUÍDA**
>
> Em componentes que usam LDAP ou AD como referência de ciclo de vida sem validação server-side de credenciais (por exemplo, sem Kerberos), desabilitar uma conta no diretório não revoga automaticamente credenciais já distribuídas. A revogação é operacional: remover os Secrets dos workloads, reiniciá-los e, quando necessário, bloquear o acesso por NetworkPolicy ou regras de rede.

© Copyright Tecnisys 2026. All rights reserved.

Consulte a página de segurança do componente para confirmar:

- se LDAP, AD ou SSO são suportados;
- quais parâmetros devem ser usados;
- quais Secrets precisam existir;
- se há overlay Helm específico;
- se a autenticação se aplica à aplicação inteira ou apenas a uma interface web.

Overlay Helm para autenticação

Alguns componentes podem disponibilizar arquivos de valores adicionais para autenticação, como overlays LDAP.

Quando documentado pelo componente, aplique o overlay junto ao arquivo principal, usando `-f <VALUES_FILE>` e `-f <LDAP_VALUES_FILE>` no comando Helm do componente.

> **NOTE**
>
> Use overlays apenas quando eles forem documentados pelo componente. Nem todos os componentes usam esse padrão.

TLS e certificados

A configuração de TLS pode ocorrer em camadas diferentes. Essas camadas não devem ser confundidas.

TLS no Ingress ou Gateway API

Quando um componente é exposto por Ingress ou Gateway API, o TLS pode ser terminado no Ingress Controller ou no Gateway.

Esse tipo de TLS protege o acesso externo ao serviço, por exemplo:

```
https://ranger.empresa.com.br
```

A configuração costuma envolver hostname, Secret TLS, certificado e parâmetros de Ingress ou Gateway API.

Para detalhes, consulte Configuração de Ingress.

© Copyright Tecnisys 2026. All rights reserved.

Rate limiting no Ingress

Além do TLS, configure limites de requisição para proteger endpoints expostos externamente contra uso abusivo. Ingress Controllers como NGINX suportam anotações específicas para esse controle:

```
annotations:
  nginx.ingress.kubernetes.io/limit-rps: "100"
  nginx.ingress.kubernetes.io/limit-connections: "10"
```

Consulte a documentação do Ingress Controller em uso para as anotações disponíveis. Os valores adequados dependem do componente e do perfil de uso esperado.

TLS na aplicação

Alguns componentes também podem exigir TLS diretamente na aplicação, em listeners, APIs internas ou portas específicas.

Esse tipo de configuração depende do componente e pode envolver:

- keystores;
- truststores;
- certificados internos;
- portas específicas;
- parâmetros próprios do componente.

Consulte a página de segurança do componente para verificar se ele documenta TLS em nível de aplicação.

TLS em integrações

Algumas integrações externas também podem exigir TLS, por exemplo:

- LDAPS para integração com diretório corporativo;
- conexão segura com banco de dados;
- conexão segura com armazenamento S3-compatível;
- comunicação segura entre componentes.

Nesses casos, a configuração pode pertencer à página de segurança do componente ou à página de integração correspondente.

© Copyright Tecnisys 2026. All rights reserved.

Autorização e políticas

Autenticação e autorização são etapas diferentes.

A autenticação define quem é o usuário.
A autorização define o que esse usuário pode acessar ou executar.

Alguns componentes possuem autorização própria. Outros podem ser integrados a mecanismos externos de autorização, como o Apache Ranger, quando essa integração estiver prevista e configurada corretamente.

Quando o Ranger for usado, há dois lados que devem ser considerados:

Lado	O que significa
Ranger Admin	Cadastro de serviços, políticas, usuários e grupos
Componente integrado	Configuração necessária para consultar ou aplicar políticas

O cadastro de um serviço no Ranger Admin não substitui, por si só, a configuração do componente para aplicar políticas em tempo de execução.

Para detalhes de políticas, UserSync, LDAP e integrações do Ranger, consulte Segurança - Ranger.

Credenciais para integrações externas

Alguns componentes precisam acessar serviços externos ou complementares, como bancos de dados, armazenamento de objetos, serviços S3-compatíveis ou APIs.

Essas integrações normalmente exigem credenciais específicas, que devem ser armazenadas em Kubernetes Secrets.

Exemplo genérico para credenciais S3-compatíveis:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> create secret generic <S3_SECRET_NAME> --from-
literal=access-key='<S3_ACCESS_KEY>' --from-literal=secret-key='<S3_SECRET_KEY>'
```

O modo de referência ao Secret varia conforme o componente. Consulte a página de integração ou segurança do componente antes de aplicar a configuração.

Para detalhes sobre conexões S3-compatíveis, consulte Configuração de Integração.

NetworkPolicy e controle de acesso à rede

NetworkPolicy restringe o tráfego de entrada e saída de pods em um cluster Kubernetes. Em ambientes de produção, é uma camada essencial de defesa em profundidade, complementar ao TLS e à autenticação da aplicação.

Para componentes TDP, considere:

- Permitir tráfego de entrada somente dos namespaces e ServiceAccounts que realmente precisam do serviço.
- Bloquear tráfego de saída desnecessário de pods com dados sensíveis.
- Tratar Services internos (não expostos via Ingress) com a mesma atenção que serviços externos.

> **NOTE**
>
> A aplicação de NetworkPolicy depende do plugin de rede (CNI) em uso no cluster. Verifique se o CNI do ambiente suporta e aplica NetworkPolicies antes de depender desse controle como única camada de proteção.

Consulte a página de segurança do componente para orientações sobre quais portas e origens de tráfego devem ser liberadas.

Auditoria e rastreabilidade

Habilite audit logging nos componentes que suportam esse recurso. Os logs de auditoria registram acessos, falhas de autenticação e operações sensíveis, sendo essenciais para investigação de incidentes e conformidade.

Cada componente TDP documenta, em sua página de segurança, se e como o audit logging pode ser habilitado. Direcionando esses logs para a solução de observabilidade

© Copyright Tecnisys 2026. All rights reserved.

do ambiente, é possível centralizar alertas e análises.

Princípios gerais de monitoramento:

- Monitore falhas de autenticação — volume elevado pode indicar tentativa de acesso não autorizado.
- Monitore acessos a dados sensíveis fora do horário operacional normal.
- Retenha logs de auditoria por período compatível com as políticas do ambiente.

Próximos passos

Para configurar segurança em um componente específico, acesse a página de segurança correspondente no índice de Configuração.

Use esta página como referência geral para entender os padrões de segurança do TDP Kubernetes. Os parâmetros exatos devem sempre ser confirmados na documentação específica do componente.

© Copyright Tecnisys 2026. All rights reserved.

Integrações — Padrões Gerais

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Esta página apresenta os principais padrões de integração usados pelos componentes TDP Kubernetes.

Ela não substitui as páginas específicas de integração de cada componente. Use esta página para entender os conceitos gerais e consulte a página do componente para aplicar os parâmetros corretos, nomes de Secrets, endpoints, arquivos de valores e exemplos completos.

Como usar esta página

As integrações no TDP Kubernetes conectam um componente a outro serviço da plataforma ou a um serviço externo.

Essas integrações podem envolver banco de dados, object storage, catálogos de metadados, serviços de governança, ferramentas de catálogo de dados, sistemas de mensageria, APIs externas ou outros serviços necessários ao funcionamento do componente.

Nem todo componente usa todos esses padrões. A disponibilidade de cada integração depende do chart, dos valores expostos, da versão instalada e da configuração do ambiente.

Tema	O que define	Onde detalhar
Banco de dados externo	Uso de um banco fora do chart do componente	Página de integração do componente
Object storage	Conexão com armazena-mento compatível com S3	Página de integração do componente

© Copyright Tecnisys 2026. All rights reserved.

Tema	O que define	Onde detalhar
Catálogo de metadados	Uso de catálogo para bancos, tabelas, schemas e formatos de dados	Página de integração do componente
Governança e autorização	Integração com serviços de políticas e controle de acesso	Página de integração ou segurança do componente
Catálogo de dados e lineage	Registro de metadados, ativos de dados e rastreabilidade	Página de integração do componente
Mensageria, eventos e conectores	Integração com brokers, tópicos, conectores ou pipelines	Página de integração do componente

Princípios gerais

Antes de configurar uma integração, valide:

- se o chart do componente documenta aquela integração;
- quais blocos de configuração devem ser usados;
- quais Secrets precisam existir;
- se o endpoint usado é interno ao cluster ou externo;
- se há dependência de TLS, autenticação ou autorização;
- se a configuração pertence ao componente, ao serviço integrado ou a ambos;
- se há jobs, hooks ou inicializações automáticas associados à integração.

Endpoints internos e externos

As integrações podem usar endereços internos do Kubernetes ou endereços externos ao cluster.

Endpoints internos do Kubernetes

Quando dois componentes rodam no mesmo cluster, a comunicação normalmente usa Services Kubernetes.

© Copyright Tecnisys 2026. All rights reserved.

O formato comum é:

```
<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local
```

Esse endereço é interno ao cluster e deve ser usado por Pods, jobs e serviços que precisam se comunicar entre si.

Exemplo genérico:

```
<POSTGRESQL_SERVICE>.<NAMESPACE>.svc.cluster.local
```

Substitua `<SERVICE_NAME>` e `<NAMESPACE>` pelos nomes reais do ambiente.

Endpoints externos

Quando o serviço integrado está fora do cluster, use o hostname ou endpoint externo correspondente.

Exemplos:

```
postgresql.empresa.com.br
```

```
s3.empresa.com.br
```

A escolha entre endpoint interno e externo depende de onde o serviço está instalado e de como ele é exposto na infraestrutura.

Credenciais e Secrets

Integrações normalmente exigem credenciais, como:

- usuário e senha de banco de dados;
- access key e secret key de object storage;
- credenciais de conectores;
- senhas de bind LDAP;
- tokens de API;

© Copyright Tecnisys 2026. All rights reserved.

- certificados, keystores ou truststores.

Essas credenciais devem ser armazenadas em Kubernetes Secrets sempre que o chart permitir.

Exemplo genérico:

Terminal input

```
kubectl -n <NAMESPACE> create secret generic <APPLICATION_NAME>-credentials --
from-literal=username='...' --from-literal=password='...'
```

O modo de referenciar o Secret varia por componente. Consulte a página específica de Integração e Segurança antes de aplicar a configuração.

> **NUNCA VERSIONE SENHAS**
>
> Não inclua senhas em texto plano em arquivos de valores commitados no Git. Use Secrets ou ferramentas próprias de gestão de credenciais.

Banco de dados externo

Alguns componentes podem usar um banco de dados externo em vez de um banco interno, embutido ou provisionado junto ao chart.

Esse tipo de integração costuma exigir:

- hostname do banco;
- porta;
- nome do banco;
- usuário;
- senha;
- Secret com credenciais;
- parâmetros de criação ou reaproveitamento da base;
- validação de conectividade entre o componente e o banco.

O padrão exato de configuração varia por chart.

© Copyright Tecnisys 2026. All rights reserved.

Exemplo conceitual:

```
externalDatabase:
  enabled: true
  host: "<DATABASE_HOST>"
  port: 5432
  database: "<DATABASE_NAME>"
  existingSecret: "<DATABASE_SECRET>"
```

> **NOTE**
>
> O exemplo acima é apenas conceitual. Use o bloco documentado na página específica do componente.

Object storage compatível com S3

Alguns componentes podem ler ou gravar dados em armazenamento compatível com S3, como serviços de object storage internos ou externos ao cluster.

Esse tipo de integração pode envolver:

- endpoint S3;
- bucket;
- access key;
- secret key;
- região;
- modo path-style;
- certificados;
- configuração Hadoop/Spark S3A;
- Secret com credenciais.

Exemplo conceitual:

```
s3:
  endpoint: "https://<S3_ENDPOINT>"
  bucket: "<S3_BUCKET>"
  existingSecret: "<S3_SECRET_NAME>"
```

© Copyright Tecnisys 2026. All rights reserved.

Em alguns charts, a integração S3 pode ser uma conexão gerenciada pela aplicação. Em outros, pode aparecer como propriedades Hadoop, Spark, catálogo, storage policy ou configuração de warehouse.

Consulte a página específica do componente para confirmar o padrão usado.

Catálogo de metadados

Alguns componentes podem depender de um catálogo de metadados para localizar bancos, tabelas, schemas, formatos de dados e localizações físicas dos dados.

Esse tipo de integração normalmente envolve:

- URI do serviço de catálogo;
- protocolo de comunicação;
- namespace e Service Kubernetes;
- configurações de warehouse;
- integração com object storage, quando os dados ficam em S3 ou equivalente;
- propriedades específicas do engine ou framework usado pelo componente.

Exemplo conceitual de URI interna:

```
thrift://<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local:<SERVICE_PORT>
```

O modo de declarar essa integração varia conforme o componente.

Governança, autorização e políticas

Alguns componentes podem ser integrados a mecanismos de governança ou autorização.

Nesses casos, é importante separar dois lados da configuração:

Lado	O que significa
Serviço de governança	Cadastro de serviços, políticas, usuários, grupos ou permissões

© Copyright Tecnisys 2026. All rights reserved.

Lado	O que significa
Componente integrado	Configuração necessária para consultar ou aplicar políticas em tempo de execução

Cadastrar um serviço em uma ferramenta de governança não garante, por si só, que o componente já esteja aplicando políticas.

Quando esse tipo de integração existir, valide:

- se o serviço foi cadastrado corretamente;
- se as políticas foram criadas ou declaradas;
- se usuários e grupos existem;
- se o componente integrado está configurado para usar o mecanismo de autorização;
- se certificados, endpoints e credenciais estão corretos.

Os detalhes devem ser consultados na página específica do componente e, quando aplicável, na página de integração ou segurança do serviço de governança.

Catálogo de dados e lineage

Algumas integrações têm como objetivo catalogar metadados, registrar ativos de dados ou rastrear lineage.

Esse tipo de integração pode envolver:

- conectores configurados na ferramenta de catálogo;
- credenciais para acessar os serviços de origem;
- permissões de leitura;
- DAGs, jobs ou pipelines de ingestão;
- endpoints internos ou externos dos serviços integrados.

A configuração pode ser feita pelo chart, pela interface da ferramenta, por workflows próprios de ingestão ou por automações específicas do componente.

Consulte a página de integração do componente responsável pelo catálogo para entender o fluxo recomendado.

© Copyright Tecnisys 2026. All rights reserved.

Mensageria, eventos e conectores

Integrações com mensageria ou conectores normalmente envolvem brokers, tópicos, conectores, credenciais e endpoints de bootstrap.

Esse tipo de configuração pode servir para:

- ingestão de dados;
- publicação de eventos;
- captura de mudanças;
- integração entre sistemas;
- pipelines de streaming.

Valide sempre:

- endpoint de bootstrap;
- protocolo de segurança;
- credenciais;
- certificados, quando houver TLS;
- permissões de leitura e escrita;
- configurações específicas do conector.

Os parâmetros concretos devem ser aplicados conforme a página de integração do componente.

Fluxo de integração

Algumas integrações exigem configuração em mais de um lugar.

Situação	Atenção
Serviço cadastrado em uma ferramenta externa	O componente ainda pode precisar ser configurado para usar esse serviço
Secret criado no Kubernetes	O chart ainda precisa referenciar esse Secret corretamente
Endpoint informado na configuração	O serviço precisa estar acessível a partir do Pod ou job que usará a integração

© Copyright Tecnisys 2026. All rights reserved.

Situação	Atenção
Política criada em um serviço de governança	O componente precisa estar preparado para consultar ou aplicar essa política
Catálogo configurado	O engine ou aplicação precisa usar esse catálogo em tempo de execução

Por isso, valide sempre o fluxo completo da integração, não apenas a criação de um recurso isolado.

Validação de uma integração

Depois de configurar uma integração, valide em três níveis.

Nível	O que validar
Configuração	Os valores do chart apontam para endpoints, Secrets e parâmetros corretos
Conectividade	O componente consegue alcançar o serviço integrado
Funcionalidade	A integração executa a operação esperada, como autenticar, consultar, gravar, catalogar ou aplicar política

Comandos úteis dependem do componente, mas normalmente incluem:

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

Terminal input

```
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl logs -n <NAMESPACE> <POD_NAME>
```

Terminal input

```
kubectl get secret -n <NAMESPACE>
```

Terminal input

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- sh -c '...'
```

Use a página específica do componente para os comandos de validação completos.

Resolução de problemas

Sintoma	Possível causa
Componente não conecta ao serviço integrado	Endpoint, porta, namespace ou DNS incorreto
Erro de autenticação	Secret ausente, credencial incorreta ou usuário sem permissão
Erro de certificado	Certificado ausente, truststore incorreta ou hostname fora do certificado
Integração criada, mas sem efeito em tempo de execução	Apenas um lado da integração foi configurado
Job de integração falha	Serviço integrado indisponível, credenciais incorretas ou parâmetros incompletos
Recurso não aparece na ferramenta de catálogo	Conector não executado, credencial sem permissão ou origem inacessível

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Possível causa
Configuração aparentemente correta, mas sem efeito	O componente pode exigir configuração complementar em outro bloco ou página

Próximos passos

Para configurar uma integração concreta, acesse a página de integração do componente correspondente.

Use esta página como referência geral para entender os padrões de integração do TDP Kubernetes. Os parâmetros, exemplos e validações devem sempre ser confirmados na documentação específica do componente.

© Copyright Tecnisys 2026. All rights reserved.

Airflow

CHART

Version 3.0.1 | Type application | AppVersion 3.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | External DB suportado | Ingress suportado | Gateway API suportado

O que é o Apache Airflow?

O Apache Airflow é a ferramenta de **orquestração de workflows** do TDP. Ele permite criar, agendar e monitorar pipelines de dados definidos como código Python (DAGs), com `KubernetesExecutor` como executor padrão.

O chart `tdp-airflow` empacota o Apache Airflow 3.0.2 para Kubernetes.

Para uma visão completa da ferramenta, sua arquitetura e funcionamento, consulte Apache Airflow — Conceitos.

Estrutura de valores (Helm)

Este é um chart umbrella. Todas as opções do Apache Airflow devem ficar agrupadas sob a chave **`tdp-airflow:`** (alias da dependência no `Chart.yaml`).

Respeitar esse prefixo garante que o Helm aplique os valores no subchart correto. Para quem opera com GitOps, isso se reflete diretamente no que o Argo CD aplica — um `argocd app diff` ou `helm template` só mostrará CPU, conexões e demais chaves se estiverem aninhadas sob `tdp-airflow:`.

```
tdp-airflow:
  apiServer:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    resources:
      requests:
        cpu: "500m"
        memory: "1Gi"
```

Visão geral

Propriedade	Valor
Chart	`tdp-airflow`
Versão do Airflow	3.0.2
Versão do chart	3.0.1
Executor padrão	`KubernetesExecutor`
Banco de metadados	PostgreSQL embutido por padrão, com opção de PostgreSQL externo
Persistência	DAGs, logs e Triggerer podem usar PVCs
Exposição HTTP	UI/API via Service interno, Ingress ou Gateway API

Páginas relacionadas

- Integrações — Airflow — PostgreSQL externo e conexão S3
- Segurança — Airflow — LDAP e Secrets
- Exposição externa: Ingress e Gateway API — exposição HTTP/HTTPS da UI

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível quando a persistência de DAGs, logs ou Triggerer estiver habilitada.

© Copyright Tecnisys 2026. All rights reserved.

Instalação

Instalação mínima via registry OCI:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow -n <NAMESPACE> --create-
namespace --timeout 10m
```

O arquivo de values adicional — por exemplo `meu-values.yaml` — é onde ficam apenas os ajustes necessários para o ambiente, sem alterar o procedimento básico:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow -n <NAMESPACE> --create-
namespace -f <VALUES_FILE> --wait --timeout 15m
```

> **DICA**
>
> A primeira instalação pode levar alguns minutos (imagens, migrations, hooks). Os flags `--wait --timeout 15m` aguardam a conclusão dos pods e hooks antes de retornar. Aumente o timeout se o seu cluster for mais lento.

OpenShift

Desde a release **3.0.1**, o Airflow é compatível com Red Hat OpenShift 4.19+.

Instalação com ajuste exigido pelas Security Context Constraints do OpenShift:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow -n <NAMESPACE> --create-
```

© Copyright Tecnisys 2026. All rights reserved.

```
namespace --timeout 10m --set tdp-
airflow.securityContexts.containers.runAsNonRoot=true
```

O parâmetro `runAsNonRoot=true` é exigido pelas Security Context Constraints (SCCs) do OpenShift.

Para detalhes sobre compatibilidade e ajustes adicionais, consulte Configuração Geral, aba **Decisões de ambiente que afetam a configuração**, no tópico **Compatibilidade do orquestrador e ajustes de segurança**.

Parâmetros principais

Parâmetros principais

A tabela a seguir resume os parâmetros mais consultados durante a configuração do Airflow. Para uma primeira revisão, os pontos que normalmente merecem mais atenção são: banco de dados, persistência de DAGs e logs, autenticação LDAP e integrações com serviços compartilhados da plataforma.

Parâmetro	Descrição	Valor por omissão
`tdp-airflow.enabled`	Habilita o deploy do Airflow	`true`
`tdp-airflow.executor`	Executor	`KubernetesExecutor`
`tdp-airflow.config.core.default_timezone`	Fuso padrão	`America/Sao_Paulo`
`tdp-airflow.apiServer.service.type`	Tipo do Service da interface web do Airflow	`ClusterIP`
`tdp-airflow.apiServer.service.ports`	Portas do Service da UI/API	Ver `helm show values`
`tdp-airflow.postgresql.enabled`	PostgreSQL embutido	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor por omissão
`tdp-airflow.dags.persistence.enabled`	PVC de DAGs	`true`
`tdp-airflow.dags.persistence.size`	Tamanho do PVC de DAGs	`5Gi`
`tdp-airflow.dags.persistence.storageClassName`	StorageClass do PVC de DAGs	`""`
`tdp-airflow.logs.persistence.enabled`	PVC de logs	`false`
`tdp-airflow.logs.persistence.size`	Tamanho do PVC de logs	`5Gi`
`tdp-airflow.triggerer.persistence.enabled`	PVC do Triggerer	`true`
`tdp-airflow.triggerer.persistence.size`	Tamanho do PVC do Triggerer	`5Gi`
`tdp-airflow.ldap.enabled`	LDAP	`false`
`tdp-airflow.ldap.apiServerConfig`	Trecho Flask-AppBuilder	Ver `helm show values`
`tdp-airflow.extraEnv`	Variáveis extras (ex.: Secret)	`""`
`tdp-airflow.data`	Metadados / DB externo	Desligado por padrão
`TDP-Settings.externalDatabase.enabled`	Helpers DB externo TDP	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor por omissão
`TDP-Settings.externalDatabase.recreate`	Recriar DB/usuário no install/upgrade	`false`
`TDP-Settings.externalDatabase.externalSecret.releaseName`	Release do Secret PostgreSQL externo	`<POSTGRESQL_RELEASE>`
`TDP-Settings.s3Connection.enabled`	Secret de conexão S3 TDP	`false`
`TDP-Settings.s3Connection.secretName`	Nome do Secret S3	`tdp-airflow-minio-connection`
`TDP-Settings.s3Connection.uri`	URI do endpoint S3	`https://<S3_ENDPOINT>`
`TDP-Settings.gateway.ingress.enabled`	Habilita Ingress	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita Gateway API	`false`

Detalhes de configuração

Configuração padrão

Por padrão, o chart prioriza uma instalação funcional com o mínimo de dependências externas: executor em Kubernetes, metadados em PostgreSQL local e DAGs em volume persistente. A revisão desses padrões depende da política operacional adotada para banco, armazenamento, autenticação e observabilidade, conforme descrito em Configuração Geral.

- **Executor**: `KubernetesExecutor`

© Copyright Tecnisys 2026. All rights reserved.



- **Banco de dados**: PostgreSQL embutido (subchart), com `tdp-airflow.postgresql.enabled: true` por padrão
- **DAGs**: persistência em PVC **habilitada** por padrão (`tdp-airflow.dags.persistence.enabled: true`, tamanho típico `5Gi`)
- **Logs**: persistência em PVC **desabilitada** por padrão (`tdp-airflow.logs.persistence.enabled: false`); habilite apenas se o StorageClass atender ao padrão de acesso necessário (em geral RWX)

Configuração do banco de dados

O Airflow precisa de um banco de dados relacional para armazenar seus metadados: DAGs registrados, histórico de execuções, conexões, variáveis e usuários.

Sem um banco estável, o Airflow não consegue manter o histórico e os metadados que tornam o serviço utilizável no dia a dia.

Por isso, a escolha entre PostgreSQL embutido e externo é uma das primeiras decisões de configuração:

- o PostgreSQL embutido simplifica a instalação inicial;
- o PostgreSQL externo costuma ser preferido quando o ambiente já possui padrões próprios de backup, disponibilidade e administração de banco.

Para entender quando usar cada um, consulte PostgreSQL interno versus externo na página de Configuração Geral.

PostgreSQL embutido (padrão)

Indicado apenas para desenvolvimento e testes:

```
tdp-airflow:
  postgresql:
    enabled: true
```

> **ATENÇÃO**
>
> Para produção, prefira PostgreSQL externo.

PostgreSQL externo

© Copyright Tecnisys 2026. All rights reserved.



Desative o banco embutido e informe os dados de conexão do PostgreSQL externo. Na prática, isso é o cenário mais comum quando o cliente já utiliza um PostgreSQL da própria plataforma ou um serviço de banco administrado separadamente.

Para os helpers de integração TDP:

```
TDP-Settings:
  externalDatabase:
    enabled: true
    recreate: false
    externalSecret:
      releaseName: "<POSTGRESQL_RELEASE>"

tdp-airflow:
  postgresql:
    enabled: false

  data:
    metadataSecretName: "<RELEASE_NAME>-airflow-database"
    metadataConnection:
      user: airflow
      pass: ""
      protocol: postgresql
      host: "<POSTGRESQL_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 5432
      db: airflow
      sslmode: disable
```

Com `TDP-Settings.externalDatabase.enabled: true`, o chart utiliza o release informado para alinhar banco, usuário e Secret de metadados à stack TDP.

> **DICA**
>
> Prefira definir `metadataSecretName` e deixar `pass` vazio quando os jobs TDP forem responsáveis por gerar o Secret de conexão.

Persistência de DAGs (PVC)

Os DAGs precisam ficar disponíveis para os componentes que orquestram e executam os fluxos. Por isso, esta seção trata menos de "armazenar arquivos" e mais de garantir

© Copyright Tecnisys 2026. All rights reserved.

que o Airflow consiga encontrar os DAGs de forma estável no ambiente.

Por padrão o chart já cria PVC para DAGs. Ajuste tamanho, StorageClass ou modo de acesso conforme o cluster:

```
tdp-airflow:
  dags:
    persistence:
      enabled: true
      size: 5Gi
      storageClassName: ""
```

> **MODO DE ACESSO**
>
> Com `KubernetesExecutor`, vários pods precisam ler os mesmos DAGs.
>
> Se o scheduler, webserver e tasks não compartilharem o volume, use uma StorageClass com **ReadWriteMany** (RWX), conforme permitido pelo ambiente.

Persistência de logs (PVC)

Por padrão os logs não ficam em PVC compartilhado. Isso simplifica a instalação inicial e evita depender de um tipo específico de armazenamento logo no primeiro uso.

Na prática, com `enabled: false`, a retenção dos logs depende da estratégia de observabilidade já adotada no ambiente. Com `enabled: true`, o cluster precisa oferecer um volume compatível com escrita por múltiplos componentes.

Padrão do chart: `enabled: false`. Se habilitar, avalie o **modo de acesso** ao volume: os logs são escritos por vários componentes e RWO costuma ser inadequado.

```
tdp-airflow:
  logs:
    persistence:
      enabled: false
      size: 5Gi
```

© Copyright Tecnisys 2026. All rights reserved.



> **ATENÇÃO**
>
> Com RWO, a persistência de logs costuma ser inadequada.
>
> Prefira RWX ou outra estratégia de logs (por exemplo, stack externa) alinhada ao ambiente.

Persistência do Triggerer (PVC)

O Triggerer é um componente de Airflow 2.2+ responsável por gerenciar eventos e execuções assíncronas de tasks. Por padrão, armazena logs de heartbeats em um PVC separado.

O padrão do chart é:

```
tdp-airflow:
  triggerer:
    persistence:
      enabled: true
      size: 5Gi
```

Ajuste o tamanho e a StorageClass conforme necessário, seguindo as mesmas recomendações de Armazenamento descrito em Configuração Geral.

Conexão S3 (TDP)

Configure esta seção quando integrações TDP precisarem de parâmetros de conexão S3-compatible. A configuração `s3Connection` cria ou referencia um Secret com esses parâmetros; ela **não substitui**, por si só, a configuração de persistência de DAGs.

O conceito usado pela plataforma está detalhado na página Configuração Geral, aba **Decisões de ambiente que afetam a configuração**, no tópico **Armazenamento S3-compatível (Ozone, MinIO ou outro endpoint)**.

Com `TDP-Settings.s3Connection.enabled: true`, o chart cria um Secret com parâmetros de conexão S3-compatible para integrações TDP. Use placeholders; não versione credenciais em texto plano:

© Copyright Tecnisys 2026. All rights reserved.

```
TDP-Settings:
  s3Connection:
    enabled: true
    secretName: "<S3_SECRET_NAME>"
    name: "<CONNECTION_NAME>"
    type: "aws"
    accessKey: "<S3_ACCESS_KEY>"
    secretKey: "<S3_SECRET_KEY>"
    uri: "https://<S3_ENDPOINT>"
```

Variáveis de ambiente adicionais

Use `tdp-airflow.extraEnv` para referenciar Secrets ou injetar outras variáveis (por exemplo, senha LDAP):

```
tdp-airflow:
  extraEnv: |
    - name: EXAMPLE_ENV
      valueFrom:
        secretKeyRef:
          name: <SECRET_NAME>
          key: <SECRET_KEY>
```

Acesso & Segurança

Acesso

O acesso à interface web do Airflow é feito por um Service Kubernetes, normalmente do tipo `ClusterIP`.

Para acesso local durante validação ou testes:

Terminal input

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME>-api-server 8080:8080
```

Para exposição externa estável por Ingress ou Gateway API, consulte Exposição externa: Ingress e Gateway API.

© Copyright Tecnisys 2026. All rights reserved.

Autenticação LDAP

LDAP é opcional e vem desligado por padrão.

Enquanto estiver desligado, não há dependência de diretório corporativo nem necessidade de Secrets adicionais para esse fim. Ao habilitar LDAP, a autenticação do Airflow passa a depender das definições do diretório da organização, como servidor, base de busca, usuários e variáveis sensíveis armazenadas em Secret.

A configuração usa `tdp-airflow.ldap` e variáveis injetadas via `tdp-airflow.extraEnv`.

> **DOCUMENTAÇÃO DETALHADA**
>
> Veja Segurança — Airflow para Secret de bind e exemplos de configuração LDAP.

Desinstalação

Desinstalação

Para remover a instalação do Airflow:

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Este comando remove todos os recursos do release (pods, services, configmaps, secrets). Os PVCs (volumes) são mantidos por padrão para preservar os dados. Se deseja remover também os PVCs:

Terminal input

```
kubectl delete pvc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.

Airflow – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 3.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado

O chart `tdp-airflow` suporta autenticação **LDAP** opcional para a interface web do Airflow (Flask-AppBuilder). Por padrão, `tdp-airflow.ldap.enabled` é `false`.

Quando habilitada, a autenticação LDAP do Airflow é configurada por `tdp-airflow.ldap` e por variáveis de ambiente injetadas via `tdp-airflow.extraEnv`. Forneça um Secret com a senha de bind LDAP e referencie esse Secret em `tdp-airflow.extraEnv`.

Pré-requisitos

- Servidor LDAP acessível a partir do cluster Kubernetes
- Credenciais de bind (usuário e senha), normalmente fornecidas via Secret

Credenciais e Secrets

Crie o Secret com a senha de bind LDAP antes de instalar o chart:

Terminal input

```
kubectl create secret generic tdp-airflow-ldap-bind \
  --from-literal=password="<LDAP_BIND_PASSWORD>" \
  -n <NAMESPACE>
```

IMPORTANTE

© Copyright Tecnisys 2026. All rights reserved.



> Se o Secret referenciado em `tdp-airflow.extraEnv` não existir, os pods podem falhar na inicialização.

Configuração LDAP

Em um arquivo de values personalizado (por exemplo `<VALUES_FILE>`), use o prefixo **`tdp-airflow.ldap.*`** e injete a senha com **`tdp-airflow.extraEnv`**:

```
tdp-airflow:
  ldap:
    enabled: true
    apiServerConfig: |
      import os
      from flask_appbuilder.security.manager import AUTH_LDAP

      AUTH_TYPE = AUTH_LDAP
      AUTH_LDAP_SERVER = "ldap://<LDAP_HOST>:389"
      AUTH_LDAP_USE_TLS = False
      AUTH_LDAP_BIND_USER = "uid=
<LDAP_BIND_USER>,cn=users,cn=accounts,dc=example,dc=com"
      AUTH_LDAP_BIND_PASSWORD = os.environ.get("LDAP_BIND_PASSWORD", "")
      AUTH_LDAP_SEARCH = "cn=users,cn=accounts,dc=example,dc=com"
      AUTH_LDAP_UID_FIELD = "uid"

  extraEnv: |
    - name: LDAP_BIND_PASSWORD
      valueFrom:
        secretKeyRef:
          name: tdp-airflow-ldap-bind
          key: password
```

> **TIP**
>
> A senha de bind não deve ficar em texto plano no `values.yaml` versionado; use `valueFrom.secretKeyRef` em `tdp-airflow.extraEnv`.

Instalação ou atualização

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

Parâmetros

Parâmetro	Descrição
`tdp-airflow.ldap.enabled`	Habilita autenticação LDAP
`tdp-airflow.ldap.apiServerConfig`	Trecho Python Flask-AppBuilder (AUTH_LDAP, etc.)
`tdp-airflow.extraEnv`	Variáveis extras; use para `LDAP_BIND_PASSWORD` via Secret

Solução de problemas

Problema	Causa provável	Solução
Pod não inicia	Secret do bind ausente ou nome incorreto	Criar/corrigir Secret e alinhar `secretKeyRef.name` em `tdp-airflow.extraEnv`
Credenciais inválidas	Bind DN ou senha incorretos	Validar bind com ferramentas LDAP do ambiente
Usuário não encontrado	Base de busca ou campo UID incorretos	Validar `AUTH_LDAP_SEARCH` e `AUTH_LDAP_UID_FIELD` conforme o diretório

© Copyright Tecnisys 2026. All rights reserved.

Airflow – Integração

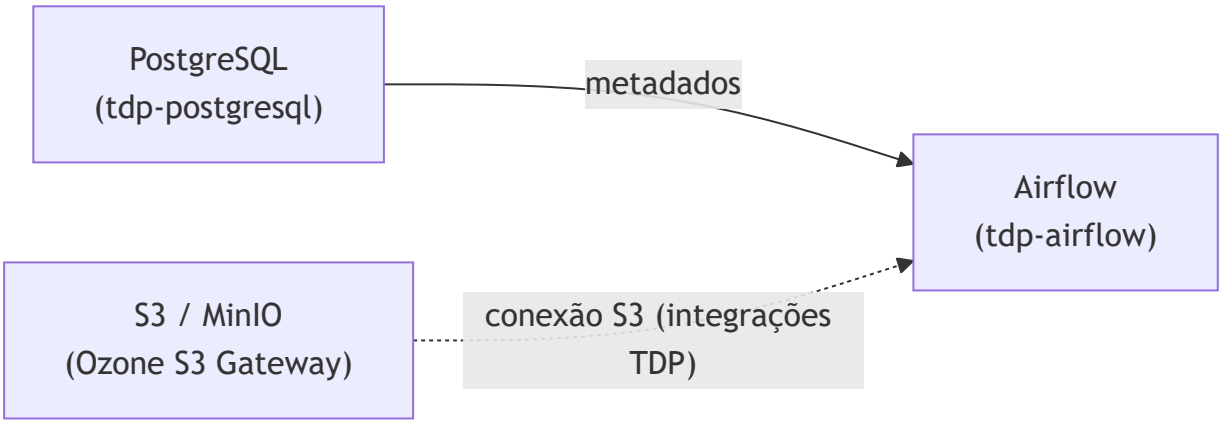
CHART

Version 3.0.1 | Type application | AppVersion 3.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O chart `tdp-airflow` suporta integração com **armazenamento S3-compatível** (Ozone S3 Gateway, MinIO ou outro endpoint S3) via `TDP-Settings.s3Connection`. A configuração do banco de dados de metadados (PostgreSQL embutido ou externo) é parte do processo de instalação e está documentada em Configuração do Airflow.

Integração com S3 / armazenamento compatível

Habilitar `TDP-Settings.s3Connection` cria um Secret com parâmetros de conexão para integrações TDP (endpoint compatível com S3). Essa configuração **não substitui**, por si só, a persistência de DAGs:

```
TDP-Settings:
  s3Connection:
    enabled: true
    secretName: "<S3_SECRET_NAME>"
    name: "<CONNECTION_NAME>"
    type: "aws"
```

© Copyright Tecnisys 2026. All rights reserved.

```
    accessKey: "<S3_ACCESS_KEY>"
    secretKey: "<S3_SECRET_KEY>"
    uri: "https://<S3_ENDPOINT>"
```

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

CREDENCIAIS

Evite commitar `accessKey` e `secretKey` em repositório. Prefira `--set` pontual, gestão de secrets no cluster ou referências via `tdp-airflow.extraEnv`, conforme a política de segurança do ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Argo CD

CHART

Version 3.0.1 | Type application | AppVersion 3.2.5

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | Ingress suportado | Gateway API suportado

O que é o Argo CD?

O Argo CD é a ferramenta de **entrega contínua GitOps** do TDP.

Ele monitora um repositório Git e garante que o que está no cluster Kubernetes esteja sempre sincronizado com o que está no Git — se você alterar um arquivo de configuração no repositório, o Argo CD detecta a mudança e aplica no cluster automaticamente.

No TDP Kubernetes, o Argo CD é usado principalmente para gerenciar o ciclo de vida dos próprios charts TDP no modelo **App of Apps**: uma `Application` raiz que gerencia outras `Applications`, cada uma responsável por um componente TDP.

Isso é descrito em detalhes nos guias de instalação via Argo CD e atualização.

> **O ARGO CD NÃO SUBSTITUI O HELM**
>
> O Argo CD utiliza o Helm para renderizar os charts e aplicar os recursos no cluster. A diferença é que você não executa `helm upgrade` manualmente — o Argo CD faz isso automaticamente ao detectar mudanças no Git. O fluxo de configuração (arquivos de values, secrets) permanece o mesmo.

© Copyright Tecnisys 2026. All rights reserved.



O chart `tdp-argo` empacota o Argo CD 3.2.5 como wrapper sobre o chart upstream `argo-cd`.

Estrutura de valores (Helm)

Todos os valores devem ficar sob **`tdp-argo:`** (alias da dependência no `Chart.yaml`):

```
tdp-argo:
  server:
    resources:
      requests:
        cpu: "250m"
        memory: "256Mi"
```

Visão geral

Propriedade	Valor
Chart	`tdp-argo`
Versão do Argo CD	3.2.5
Versão do chart	3.0.1
Upstream	chart `argo-cd`
Modelo	App of Apps (GitOps)
Recursos suportados	LDAP, Ingress e Gateway API

Páginas relacionadas

- Segurança — Argo CD — LDAP via Dex, RBAC e NetworkPolicy
- Exposição externa: Ingress e Gateway API — exposição HTTP/HTTPS da UI
- Instalação via Argo CD — bootstrap GitOps e App of Apps
- Atualização do Argo CD — atualização das versões dos charts

Configurações TDP específicas

© Copyright Tecnisys 2026. All rights reserved.



O chart inclui configurações específicas do TDP, fora do escopo do upstream:

- `skipCrdCheck` — valida se o `tdp-crds` está instalado antes do deploy.
- `TDP-Settings.gateway` — controle de exposição via Ingress ou Gateway API.
- `gatewayApi` — configuração do Gateway API para os HTTPRoutes.

Pré-requisitos: CRDs (tdp-crds)

Obrigatório instalar os CRDs antes do chart principal:

Terminal input

```
helm upgrade --install tdp-crds \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  -n <NAMESPACE> --create-namespace
```

O chart principal mantém `tdp-argo.crds.install=false` para evitar conflitos de ciclo de vida dos CRDs no Helm. O `tdp-crds` consolida os CRDs de todos os componentes da plataforma (Argo CD, Kafka, NiFiKop, ClickHouse, OpenMetadata e cert-manager) num único release.

Instalação

Instalação

Via Script deploy.sh (Recomendado)

O script `deploy.sh` automatiza a instalação completa do Argo CD:

Terminal input

```
./deploy.sh --install -v variables.env.local
```

O que é executado:

1. `helm registry login` → autentica no OCI registry
2. `helm upgrade --install tdp-crds` → instala os CRDs
3. `helm upgrade --install tdp-argo` → instala o Argo CD

© Copyright Tecnisys 2026. All rights reserved.

4. `kubectl rollout status` → aguarda os pods prontos
5. Renderiza os templates → popula `current/`
6. Aplica os recursos comuns (AppProject, Secrets, App of Apps)

O fluxo GitOps completo (estrutura do repositório, `variables.env.local`, bootstrap e sincronização) está documentado em Instalação via Argo CD.

Via Helm Direto

Para instalação manual via Helm:

Terminal input

```
helm upgrade --install tdp-argo \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
  -n <NAMESPACE> --create-namespace \
  --set skipCrdCheck=false
```

Verificação

ConfigMap de instalação

Terminal input

```
kubectl get configmap tdp-argo-argo-install-properties -n <NAMESPACE> \
  -o jsonpath='{.data.install\.properties}' | tail -10
```

Terminal input

```
kubectl get configmap tdp-argo-argo-install-properties -n <NAMESPACE> -o
jsonpath='{.data.argo-admin-install\.properties}' | tail -10
```

Pods

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.



Todos os pods devem estar em status `Running`.

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão
`skipCrdCheck`	Pula a validação de CRDs (use apenas para `helm template`)	`false`
`tdp-argo.crds.install`	Instala CRDs via upstream (deve ser `false`)	`false`
`tdp-argo.dex.enabled`	Habilita Dex para LDAP	`false`
`tdp-argo.configs.cm.url`	URL pública do Argo CD	`https://argo.tdp.local`
`tdp-argo.configs.secret.*`	Secrets para credenciais	—
`tdp-argo.rbacConfig`	Configuração RBAC	—
`tdp-argo.global.networkPolicy.*`	Configurações de NetworkPolicy	`create: false`
`TDP-Settings.gateway.ingress.enabled`	Habilita Ingress	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita Gateway API	`false`
`tdp-argo.openshift.enabled`	Compatibilidade OpenShift	`false`

© Copyright Tecnisys 2026. All rights reserved.

Configurações principais

URL pública (`configs.cm.url`)

Alinhe a URL à forma como os usuários acessam o Argo CD (relevante para Dex/LDAP):

```
tdp-argo:
  configs:
    cm:
      url: https://<ARGOCD_HOSTNAME>
```

Application namespaces

Por padrão, o Argo CD gerencia `Applications` em **todos os namespaces** (`application.namespaces: "*"`):

```
tdp-argo:
  configs:
    params:
      application.namespaces: "*"
```

Para restringir a um conjunto específico (por exemplo, apenas `<NAMESPACE>`):

```
tdp-argo:
  configs:
    params:
      application.namespaces: "<NAMESPACE>"
```

HTTP vs HTTPS

Por padrão, o acesso HTTP é habilitado (`server.insecure: "true"`), útil atrás de Ingress, service mesh ou proxy terminador de TLS:

```
tdp-argo:
  configs:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    params:
      server.insecure: "true"
```

Para forçar HTTPS, defina o valor como `"false"`.

Compatibilidade OpenShift

Para deploy em OpenShift, habilite a compatibilidade:

```
tdp-argo:
  openshift:
    enabled: true
```

Alternativamente, é possível habilitar a compatibilidade diretamente na instalação:

Terminal input

```
helm upgrade --install tdp-argocd \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
  --namespace argocd --create-namespace \
  --set tdp-argo.openshift.enabled=true \
  --set tdp-argo.redis.securityContext.runAsUser=null \
  --set tdp-argo.redis.securityContext.fsGroup=null
```

Isso cria a RoleBinding da SCC `anyuid` para o Redis e ajusta os security contexts conforme necessário.

Dimensionamento de recursos

Configurações de recursos recomendadas por componente do Argo CD:

```
tdp-argo:
  server:
    resources:
      requests:
        cpu: "250m"
        memory: "512Mi"
      limits:
        cpu: "1"
```

© Copyright Tecnisys 2026. All rights reserved.

```
        memory: "2Gi"
  repoServer:
    replicas: 2
    resources:
      requests:
        cpu: "250m"
        memory: "256Mi"
      limits:
        cpu: "1"
        memory: "2Gi"
  controller:
    replicas: 2
    resources:
      requests:
        cpu: "500m"
        memory: "1Gi"
      limits:
        cpu: "2"
        memory: "4Gi"
  redis:
    resources:
      requests:
        cpu: "50m"
        memory: "64Mi"
      limits:
        cpu: "250m"
        memory: "512Mi"
  dex:
    resources:
      requests:
        cpu: "50m"
        memory: "64Mi"
      limits:
        cpu: "250m"
        memory: "256Mi"
```

Overlays Disponíveis

O chart inclui overlays para cenários específicos:

- `values-ldap.yaml` — Configuração LDAP pré-preenchida
- `values-openshift.yaml` — OpenShift SCC compatibilidade
- `values-tdp-gateway.yaml` — Gateway API configuração

© Copyright Tecnisys 2026. All rights reserved.



- `values-tdp-ingress.yaml` — Ingress configuração
- `values-tdp-ingress-openshift.yaml` — Ingress + OpenShift

Uso:

Terminal input

```
helm upgrade --install tdp-argo \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
  -n <NAMESPACE> \
  -f values-ldap.yaml \
  -f values-tdp-gateway.yaml
```

Acesso & Segurança

Acesso e credenciais

- **Usuário:** `admin`
- **Senha inicial:** armazenada no Secret `argocd-initial-admin-secret`:

Terminal input

```
kubectl -n <NAMESPACE> get secret argocd-initial-admin-secret \
  -o jsonpath="{.data.password}" | base64 -d
```

Altere a senha após o primeiro acesso na interface do Argo CD (**Settings** → **Account** → **Update Password**).

Segurança

O Argo CD suporta autenticação **LDAP via Dex** (`tdp-argo.dex.enabled`), com mapeamento de grupos LDAP para roles em `tdp-argo.rbacConfig`. Para a configuração completa — criação do Secret de bind, `dex.config`, política RBAC e solução de problemas — consulte Segurança — Argo CD.

Exposição externa

A UI/API do Argo CD pode ser exposta via **Ingress** ou **Gateway API** (mutuamente exclusivos), controlados por `TDP-Settings.gateway`. Para os exemplos completos de

© Copyright Tecnisys 2026. All rights reserved.

Ingress e Gateway API, consulte Exposição externa.

NetworkPolicy

Por padrão, o chart não cria NetworkPolicies (`tdp-argo.global.networkPolicy.create: false`), mantendo o tráfego liberado. Para habilitar as policies allow-all do upstream ou aplicar uma restrição de rede própria, consulte Segurança — Argo CD.

Desinstalação

Desinstalação

Terminal input

```
helm -n <NAMESPACE> uninstall tdp-argo
```

Remove todos os recursos do release. Limpeza adicional opcional: remova PVC, ConfigMap ou Secret órfãos se a política do cluster o exigir:

Terminal input

```
kubectl delete pvc -n <NAMESPACE> --all
kubectl delete configmap -n <NAMESPACE> --all
kubectl delete secret -n <NAMESPACE> --all
```

Atualização

Para atualizar o Argo CD:

Terminal input

```
# Via deploy.sh
./deploy.sh --install -v variables.env.local

# Via Helm direto
helm upgrade tdp-argo \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  -n <NAMESPACE> \
  --set skipCrdCheck=false
```

Para o procedimento completo de atualização, consulte Atualização do Argo CD.

© Copyright Tecnisys 2026. All rights reserved.

Argo CD – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 3.2.5

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado

O chart `tdp-argo` suporta autenticação **LDAP via Dex**. LDAP vem desabilitado por padrão; habilite explicitamente editando `values.yaml` e definindo `tdp-argo.dex.enabled: true`.

Quando habilitado, o LDAP usa credenciais de bind em **`argocd-secret`** (chaves `dex.ldap.bindDN` e `dex.ldap.bindPW`), **`dex.config`** em `tdp-argo.configs.cm` e **RBAC** em `tdp-argo.rbacConfig`.

Pré-requisitos

- Servidor LDAP acessível a partir do cluster Kubernetes.
- Credenciais de bind LDAP para o Dex.
- `argocd-secret` criado manualmente com `kubectl create secret` **antes** do deploy, com as chaves `dex.ldap.bindDN` e `dex.ldap.bindPW`.
- Dex habilitado explicitamente com `tdp-argo.dex.enabled: true`.
- `tdp-argo.configs.cm.dex.config` preenchido com host LDAP e `baseDN`.

Credenciais e Secrets

Crie o `argocd-secret` **antes** de instalar o chart, para que o Dex leia as credenciais de bind via referências `$dex.ldap.bindDN` / `$dex.ldap.bindPW` e o Helm não tente gerenciá-lo:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl create namespace <NAMESPACE>

kubectl create secret generic argocd-secret \
  --from-literal=dex.ldap.bindDN="uid=<LDAP_BIND_USER>,cn=users,cn=accounts,dc=
<LDAP_DOMAIN>,dc=com" \
  --from-literal=dex.ldap.bindPW="<LDAP_BIND_PASSWORD>" \
  -n <NAMESPACE>
```

Em seguida, mantenha `createSecret: false` no arquivo de valores para que o Helm não gerencie o Secret:

```
tdp-argo:
  configs:
    secret:
      createSecret: false
```

> **DANGER**
>
> Não commite senhas. Use arquivos de valores privados, CI seguro ou gestão de secrets externa.

Configuração LDAP

```
tdp-argo:
  configs:
    cm:
      url: https://<ARGOCD_HOST>
      dex.config: |
        connectors:
        - type: ldap
          id: ldap
          name: LDAP
          config:
            host: "<LDAP_HOST>:389"
            insecureNoSSL: true
            insecureSkipVerify: true
            bindDN: "$dex.ldap.bindDN"
            bindPW: "$dex.ldap.bindPW"
            userSearch:
```

© Copyright Tecnisys 2026. All rights reserved.

```
              baseDN: "cn=users,cn=accounts,dc=example,dc=com"
              filter: "(objectClass=person)"
              username: uid
              idAttr: uid
              emailAttr: mail
              nameAttr: givenName
    secret:
      createSecret: false

  dex:
    enabled: true
```

Ajuste `host`, `baseDN` e flags SSL conforme seu diretório. Quando `createSecret: false`, crie e mantenha o `argocd-secret` por outro meio, com as chaves referenciadas em `dex.config`.

O `values.yaml` padrão mantém Dex desabilitado e deixa `dex.config` vazio. Nesse estado, a página de login não mostra o botão LDAP/SSO.

> **PRODUÇÃO: PREFIRA LDAPS**
>
> O exemplo acima usa LDAP em texto claro (`<LDAP_HOST>:389` com `insecureNoSSL: true` e `insecureSkipVerify: true`), adequado apenas para ambientes de teste. Em produção, use **LDAPS** (porta 636): defina `insecureNoSSL: false` e `insecureSkipVerify: false`, ajuste `host` para `<LDAP_HOST>:636` e disponibilize a CA do servidor LDAP ao Dex.

Autorização e políticas

```
tdp-argo:
  rbacConfig:
    policy.default: role:readonly
    policy.csv: |
      g:devops-admins, role:admin
      g:data-platform, role:admin
      g:data-read, role:readonly
      p, role:ldap-user, applications, *, default/<NAMESPACE>, get
      p, role:ldap-user, applications, *, default/<NAMESPACE>, sync
      p, role:ldap-user, applications, *, default/<NAMESPACE>, override
```

© Copyright Tecnisys 2026. All rights reserved.



```
      p, role:ldap-user, clusters, *, *, get
      g, *, role:ldap-user
```

Substitua `<NAMESPACE>` pelo projeto/namespace lógico do Argo CD que você utiliza.

Este bloco já vem ativo por padrão no chart, mesmo sem LDAP/Dex habilitado.
Os grupos `devops-admins`, `data-platform` e `data-read` e o `<NAMESPACE>` são valores de exemplo: substitua-os pelos grupos reais do seu diretório LDAP e pelo namespace do seu projeto, caso contrário essas regras não se aplicam a nenhum usuário.

O bloco `rbacConfig.policy.csv` combina três tipos de regra:

- **Mapeamento de grupos**: linhas como `g:devops-admins, role:admin` associam grupos LDAP a roles internas do Argo CD.
- **Permissões de role customizada**: linhas `p, role:ldap-user, ...` definem o que a role `ldap-user` pode executar.
- **Atribuição padrão**: `g, *, role:ldap-user` associa usuários autenticados à role `ldap-user` por padrão.

NetworkPolicy

Por padrão, o chart não cria NetworkPolicies (`tdp-argo.global.networkPolicy.create: false`), mantendo o tráfego liberado (allow-all).

Para habilitar as policies allow-all do upstream:

```
tdp-argo:
  global:
    networkPolicy:
      create: true
      defaultDenyIngress: false
```

Os templates do upstream não permitem customizar as regras via valores Helm. Para uma restrição real, aplique uma `NetworkPolicy` própria — por exemplo, permitindo ingresso aos pods do Argo CD apenas a partir de um namespace confiável (`<NAMESPACE>` é o namespace onde o Argo CD roda; `<TRUSTED_NAMESPACE>`, o namespace de origem autorizado):

© Copyright Tecnisys 2026. All rights reserved.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: argocd-restrict-ingress
  namespace: <NAMESPACE>
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchExpressions:
              - key: kubernetes.io/metadata.name
                operator: In
                values:
                  - <TRUSTED_NAMESPACE>
```

Instalação ou atualização

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-argo \
  -n <NAMESPACE> --create-namespace
```

Depois do deploy ou upgrade, confirme que o Dex está em execução:

Terminal input

```
kubectl get pods -n <NAMESPACE> | grep dex
kubectl get svc -n <NAMESPACE> | grep dex
```

Confirme também os recursos RBAC criados pelo chart:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl get serviceaccount -n <NAMESPACE> | grep argo
kubectl get role -n <NAMESPACE> | grep argo
kubectl get rolebinding -n <NAMESPACE> | grep argo
```

Login

- **UI:** acesse `https://<ARGOCD_HOST>` e use **"LOG IN VIA LDAP"** (login/senha local é só para contas locais).
- **CLI:** `argocd login <ARGOCD_HOST> --sso`

Solução de problemas

Verificações iniciais

Verifique os logs do Argo CD server para erros de conexão LDAP e avaliação RBAC:

Terminal input

```
kubectl logs -n <NAMESPACE> deployment/<DEPLOYMENT_NAME> -f
```

Teste LDAP a partir do pod Dex (exemplo com `ldapsearch`):

Terminal input

```
kubectl exec -n <NAMESPACE> deployment/<DEPLOYMENT_NAME> -- \
  ldapsearch -x -H ldap://<LDAP_HOST>:389 \
  -D "<LDAP_BIND_DN>" \
  -w "<PASSWORD>" \
  -b "<LDAP_USER_BASE_DN>" \
  "(uid=<USERNAME>)"
```

Substitua nomes dos deployments pelos recursos reais (`kubectl get deploy -n <NAMESPACE>`).

Problemas comuns

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	O que verificar
`no such host` para `argocd-dex-server`	Dex desabilitado ou Service do Dex não criado	Definir `tdp-argo.dex.enabled: true` e confirmar `kubectl get pods,svc -n <NAMESPACE>`
`config referenced key does not exist in secret`	`dex.config` referencia chaves que não existem no `argocd-secret`	Confirmar que o Secret `argocd-secret` existe e contém `dex.ldap.bindDN` e `dex.ldap.bindPW`
`Invalid username or password` ou LDAP inválido	Falha de conectividade, bind DN/senha incorretos ou busca de usuário incorreta	Validar acesso ao servidor LDAP a partir do cluster, conferir `bindDN`/senha, `userSearch.baseDN`, `filter`, e logs do servidor LDAP
Usuário autenticado sem permissão	Usuário/grupo LDAP não mapeado para uma role do Argo CD	Revisar `tdp-argo.rbacConfig.policy.csv`, confirmar nomes dos grupos LDAP e verificar mensagens de avaliação RBAC nos logs do Argo CD

© Copyright Tecnisys 2026. All rights reserved.

ClickHouse

CHART

Version 3.0.1 | Type application | AppVersion 25.8.11.66

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | Ingress suportado | Gateway API suportado

O chart `tdp-clickhouse` empacota o ClickHouse 25.8.11.66 para Kubernetes, utilizando o **Altinity ClickHouse Operator** (`ClickHouseInstallation`).

Esta página foca a **configuração base do componente**: instalação, parâmetros principais, persistência, recursos e um cenário de alta disponibilidade como exemplo de partida.

Para instaladores e operadores, o fluxo típico é: garantir `StorageClass` e limites de recurso adequados, aplicar o chart com valores base, validar o `ClickHouseInstallation` (CHI) e os Services, e só então avançar para HA, LDAP ou integração S3 quando o ambiente exigir.

Para detalhes de segurança e integrações, veja Segurança — ClickHouse e Integrações — ClickHouse.

O que é o ClickHouse?

O ClickHouse é um banco de dados colunar de alta performance voltado para analytics.

Em vez de armazenar dados linha por linha, ele armazena coluna por coluna. Isso favorece leituras rápidas em cenários de agregação, relatórios e consultas analíticas sobre grandes volumes de dados.

No TDP Kubernetes, o ClickHouse é normalmente usado como destino de dados já processados, sendo depois consultado por ferramentas como **Superset**, **CloudBeaver** e

© Copyright Tecnisys 2026. All rights reserved.



Trino.

> **PARA SABER MAIS**
>
> Consulte ClickHouse — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Como o ClickHouse opera no Kubernetes

No TDP Kubernetes, o **ClickHouse** é implantado com o **Altinity ClickHouse Operator**, no mesmo espírito declarativo de outros operadores da plataforma (por analogia ao Kafka com Strimzi): em vez de criar manualmente *Pods* e *Services*, declara-se o estado desejado num recurso `ClickHouseInstallation` (CHI), e o operador reconcilia esse estado com o cluster.

Na prática, no namespace do release, os elementos típicos são:

Componente	Descrição
Altinity Operator	Gerencia o ciclo de vida do ClickHouseInstallation
ClickHouseInstallation	Recurso que define shards, réplicas, storage e configuração do cluster
Pods do ClickHouse	Pods de dados criados a partir do CHI
Services	Endpoints de acesso ao ClickHouse
PVCs	Volumes persistentes para os dados
Jobs opcionais	Recursos auxiliares para TDP user, S3 e cleanup

Estrutura de valores (Helm)

O chart `tdp-clickhouse` distribui a configuração nos blocos abaixo:

© Copyright Tecnisys 2026. All rights reserved.



- `tdp-clickhouse:` — configuração principal do componente, incluindo `clickhouse.*`, `operator.*`, `keeper.*` e `cleanup.*`.
- `TDP-Settings:` — configurações TDP opcionais para integração S3 e controle de exposição via Ingress ou Gateway API.
- `ingress:` e `gatewayApi:` — parâmetros usados quando a exposição externa é habilitada.

```
tdp-clickhouse:
  clickhouse:
    enabled: true
  operator:
    enabled: true
  cleanup:
    enabled: true

TDP-Settings:
  s3Connection:
    enabled: false
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: false

ingress:
  hosts:
    - host: clickhouse.tdp.local

gatewayApi:
  clickhouse:
    enabled: true
```

Visão geral

Propriedade	Valor
Chart	`tdp-clickhouse`
Versão do ClickHouse	25.8.11.66

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Versão do chart	3.0.1
Operador	Altinity ClickHouse Operator
Recurso principal	`ClickHouseInstallation`
Persistência	PVCs para os dados do ClickHouse e, em HA, para o ClickHouse Keeper
Exposição HTTP	Service interno, Ingress ou Gateway API para a interface HTTP

Páginas relacionadas

- Integrações — ClickHouse — S3/Ozone, usuários TDP e integrações de consumo
- Segurança — ClickHouse — LDAP, Secrets, TLS e permissões
- Exposição externa: Ingress e Gateway API — exposição HTTP/HTTPS da interface

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- `StorageClass` padrão no cluster ou `tdp-clickhouse.clickhouse.persistence.storageClass` definido explicitamente.

Instalação

Instalação (OCI)

O `helm upgrade --install` é idempotente: na primeira execução cria o release; nas seguintes aplica apenas o diff de valores. Use sempre o registry de produção indicado e um arquivo de valores versionado (sem credenciais em texto claro) para reproduzir o mesmo comportamento entre ambientes.

Use o registry de produção:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> --create-namespace
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão (referência)
`tdp-clickhouse.clickhouse.enabled`	Habilitar cluster	`true`
`tdp-clickhouse.clickhouse.image.repository`	Repositório da imagem do ClickHouse	`clickhouse/clickhouse-server`
`tdp-clickhouse.clickhouse.image.tag`	Tag da imagem do ClickHouse	25.8.11.66
`tdp-clickhouse.clickhouse.persistence.*`	PVC: tamanho, `storageClass` e `accessModes`	execute `helm show values`
`tdp-clickhouse.clickhouse.resources`	Requests e limits de CPU/memória	execute `helm show values`
`tdp-clickhouse.clickhouse.defaultUser.password`	Senha do usuário `default`	`""`
`tdp-clickhouse.clickhouse.extraUsers`	Perfis e usuários adicionais via XML	execute `helm show values`
`tdp-clickhouse.clickhouse.tdpUser.enabled`	Usuário TDP com job de grants	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão (referência)
`tdp-clickhouse.clickhouse.tdpUser.password`	Senha do usuário TDP (definir via values override privado quando necessário)	`""`
`tdp-clickhouse.operator.enabled`	Habilitar o Altinity Operator	`true`
`tdp-clickhouse.cleanup.enabled`	Habilitar job de cleanup pré-delete	`true`
`TDP-Settings.s3Connection.enabled`	Habilitar job de criação de bucket e Secret S3	`false`
`TDP-Settings.s3Connection.bucket`	Nome do bucket S3	`""`
`TDP-Settings.s3Connection.accessKey`	Chave de acesso S3	`""`
`TDP-Settings.s3Connection.secretKey`	Chave secreta S3	`""`
`TDP-Settings.s3Connection.uri`	URI do endpoint S3	`""`
`TDP-Settings.s3Connection.selfSigned`	Ignorar verificação TLS para endpoints autoassinados	`false`

Detalhes de configuração

Labels comuns

© Copyright Tecnisys 2026. All rights reserved.



`tdp-clickhouse.commonLabels` permite aplicar labels adicionais aos recursos criados pelo chart. É útil quando o cluster usa convenções próprias de inventário, observabilidade ou governança.

Operador

O parâmetro `tdp-clickhouse.operator.enabled` controla a instalação do Altinity Operator como dependência do chart. Na maioria dos ambientes, o padrão `true` é suficiente.

Desabilitar o operador só faz sentido quando você já possui uma instalação do Altinity no cluster compatível com o chart e deseja reutilizá-la — caso contrário, o CHI não será reconciliado e os pods de ClickHouse não aparecerão como esperado. Em dúvida, mantenha `true` e trate o operador como pré-requisito implícito da stack TDP.

Logging

O chart substitui o arquivo de configuração de logging do ClickHouse via `tdp-clickhouse.operator.configs.configdFiles.01-clickhouse-02-logger.xml`.

> **NÍVEL DE LOG**
>
> Use sempre `<level>information</level>` — **não** `<level>info</level>`. O valor `info` não é reconhecido pelo ClickHouse/Poco e pode causar comportamento inesperado no logging.

ClickHouse: persistência e recursos

Os parâmetros abaixo costumam ser os primeiros ajustados por quem vai instalar o componente:

Persistência em PVC significa que **dados e estado** sobrevivem ao reinício do pod; sem ela (ou com tamanho subdimensionado), você arrisca falta de espaço, throttling de disco ou perda de dados em recriações. Requests e limits de CPU/memória definem quanto o scheduler reserva e até onde o pod pode crescer: valores baixos economizam cluster mas podem causar OOM ou consultas lentas sob carga — alinhe ao volume de dados e ao padrão de consultas (analytics costuma exigir mais memória).

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-clickhouse:
  clickhouse:
    persistence:
      enabled: true
      size: 5Gi
      storageClass: ""
      accessModes: ["ReadWriteOnce"]

    resources:
      requests:
        cpu: "1"
        memory: "2Gi"
      limits:
        cpu: "2"
        memory: "4Gi"
```

> **ATENÇÃO**
>
> Use o prefixo `tdp-clickhouse.clickhouse.resources` e `tdp-clickhouse.clickhouse.persistence`. Não use `tdp-clickhouse.resources` isolado.

StorageClass Portworx

Quando o cluster usa Portworx para volumes de banco de dados, ajuste `tdp-clickhouse.clickhouse.persistence.storageClass` e as annotations de volume conforme o padrão operacional do cluster:

```
tdp-clickhouse:
  clickhouse:
    persistence:
      enabled: true
      size: 10Gi
      storageClass: "px-db"
      accessModes: ["ReadWriteOnce"]
      annotations:
        portworx.io/repl: "3"
        portworx.io/io_priority: "high"
```

© Copyright Tecnisys 2026. All rights reserved.



```
        portworx.io/snapshot: "true"
        portworx.io/block_size: "64"
```

Usuário TDP opcional

Quando `tdp-clickhouse.clickhouse.tdpUser.enabled: true`, o chart cria um Secret (`tdp-clickhouse-tdp-user`) e executa um Job pós-install/pós-upgrade para aplicar grants.

Isso é útil quando você quer evitar o uso do usuário `default` por aplicações do ecossistema TDP.

O usuário `default` continua existindo para cenários legados ou administrativos; habilitar o usuário TDP centraliza **permissões e convenção de nome** para ferramentas que se conectam ao ClickHouse no TDP (por exemplo Superset ou CloudBeaver quando configurados para `tdpuser`), reduzindo a necessidade de compartilhar a conta administrativa padrão.

Alta disponibilidade (HA)

O chart suporta clusters ClickHouse com múltiplos **shards** e **réplicas** gerenciados pelo **ClickHouse Keeper**.

A topologia de referência apresentada abaixo é um **exemplo de partida** para ambientes com HA: **2 shards × 2 réplicas = 4 pods de dados**.

> **EXEMPLO DE PARTIDA**
>
> O exemplo de HA desta página não é obrigatório nem deve ser copiado cegamente para todos os ambientes. Use-o como base para adaptar recursos, storage, topologia e políticas de acesso à sua realidade.

Ao adaptar o exemplo, ajuste em conjunto **shards**, **réplicas**, tamanho de PVC por pod, `antiAffinity` (se há nós suficientes para dispersão) e `keeper.replicaCount` (quórum ímpar, 1 ou 3).

Conceitos fundamentais

© Copyright Tecnisys 2026. All rights reserved.

Conceito	O que faz	Quando usar
Shard	Divide os dados horizontalmente entre nós	Quando um único nó deixa de ser suficiente para o volume de dados
Réplica	Mantém uma cópia idêntica do shard	Quando é necessário tolerância a falhas e continuidade de leitura
ClickHouse Keeper	Coordena a replicação entre os pods	Obrigatório quando `replicasCount > 1`

> **TABELAS DEVEM SER REPLICATED**
>
> Para aproveitar a replicação, as tabelas precisam ser criadas com engine `ReplicatedMergeTree` ou variante equivalente. Tabelas `MergeTree` simples não são replicadas automaticamente.

Topologia 2×2 (4 pods)

```
Shard 1 ──┬── Réplica 1  (pod: chi-<RELEASE_NAME>-tdp-clickhouse-0-0)
          └── Réplica 2  (pod: chi-<RELEASE_NAME>-tdp-clickhouse-0-1)

Shard 2 ──┬── Réplica 1  (pod: chi-<RELEASE_NAME>-tdp-clickhouse-1-0)
          └── Réplica 2  (pod: chi-<RELEASE_NAME>-tdp-clickhouse-1-1)

ClickHouse Keeper ── (pod: tdp-clickhouse-keeper-0)
```

Cada par de réplicas sincroniza dados via Keeper. A perda de um pod não interrompe o acesso ao shard correspondente, desde que a tabela esteja configurada com engine replicada.

Exemplo de `values-ha.yaml`

Crie um arquivo separado para HA, por exemplo `values-ha.yaml`, para não misturar esse cenário com os valores base do componente.

© Copyright Tecnisys 2026. All rights reserved.

1. Topologia do cluster e recursos

```
tdp-clickhouse:
  clickhouse:
    antiAffinity: false   # mude para true se houver nós suficientes
    shardsCount: 2
    replicasCount: 2

    persistence:
      enabled: true
      size: 5Gi
      storageClass: ""
      accessModes: ["ReadWriteOnce"]

    resources:
      requests:
        cpu: "1"
        memory: "2Gi"
      limits:
        cpu: "2"
        memory: "4Gi"
```

2. Configuração do cluster ClickHouse

```
    configuration:
      clusters:
        - name: tdp-clickhouse
          layout:
            shardsCount: 2
            replicasCount: 2

      settings:
        distributed_ddl:
          enable: true

      macros:
        cluster: tdp-clickhouse
        shard: '{shard}'
        replica: '{replica}'
```

3. ClickHouse Keeper

© Copyright Tecnisys 2026. All rights reserved.

```
keeper:
enabled: true
replicaCount: 1   # use número ímpar: 1 ou 3

image: registry.tecnisys.com.br/community/images/clickhouse/clickhouse-keeper
tag: "25.8.11.66"

localStorage:
  size: 5Gi
  storageClass: ""
```

Para definir senha do usuário `default`, controle de acesso LDAP e boas práticas de credenciais, consulte Segurança — ClickHouse.

Aplicar a configuração HA

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> \
  -f values-ha.yaml
```

Verificação após o deploy

Após o deploy, devem aparecer os pods do ClickHouse, o recurso `chi` e, se ativado, o pod do Keeper:

Terminal input

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get chi
```

Verifique se o ClickHouse reconhece o cluster e todos os nós:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
POD=$(kubectl -n <NAMESPACE> get pods \
  -l clickhouse.altinity.com/chi=<RELEASE_NAME> \
  -o jsonpath='{.items[0].metadata.name}')
kubectl -n <NAMESPACE> exec -it "$POD" -- \
  clickhouse-client -q "SELECT cluster, shard_num, replica_num, host_name FROM
system.clusters"
```

A saída esperada deve listar os hosts do cluster configurado. No exemplo 2×2, serão 4 hosts.

Parâmetros HA principais

Parâmetro	Descrição	Padrão
`tdp-clickhouse.clickhouse.shardsCount`	Número de shards	`1`
`tdp-clickhouse.clickhouse.replicasCount`	Réplicas por shard	`1`
`tdp-clickhouse.clickhouse.antiAffinity`	Distribuir pods em nós físicos diferentes	`false`
`tdp-clickhouse.keeper.enabled`	Ativar ClickHouse Keeper	`false`
`tdp-clickhouse.keeper.replicaCount`	Número de pods Keeper	`1`
`tdp-clickhouse.keeper.image`	Imagem do Keeper	`registry.tecnisys.com.br/community/images/clickhouse/clickhouse-keeper`
`tdp-clickhouse.keeper.localStorage.size`	Storage do Keeper	`5Gi`

Integrações

© Copyright Tecnisys 2026. All rights reserved.

Para bucket S3, Secret de conexão, política de storage S3 e consumo por outras ferramentas TDP, consulte Integrações — ClickHouse.

Acesso & Segurança

Segurança

Para autenticação LDAP, gestão de senhas, controle de acesso do usuário `default` e perfis adicionais, consulte Segurança — ClickHouse.

Exposição externa (Ingress e Gateway API)

> ⚠ **CONFIGURAÇÃO OBRIGATÓRIA ANTES DE EXPOR O CLICKHOUSE EXTERNAMENTE**
>
> Por padrão, o ClickHouse só aceita conexões de `127.0.0.1/::1`. Requisições de um controlador Ingress ou Gateway são rejeitadas com `AUTHENTICATION_FAILED`.
>
> Antes de habilitar Ingress ou Gateway API, certifique-se de incluir `extraUsers` com permissão de rede aberta — os exemplos completos estão em Exposição externa: Ingress e Gateway API.

Para configuração detalhada de Ingress e Gateway API, consulte Exposição externa: Ingress e Gateway API.

Acesso

Port-forward (HTTP)

Terminal input

```
kubectl -n <NAMESPACE> port-forward svc/clickhouse-<RELEASE_NAME> 8123:8123
curl "http://localhost:8123/?query=SELECT%201"
```

`clickhouse-<RELEASE_NAME>` é o Service HTTP do ClickHouse. Confirme com `kubectl get svc -n <NAMESPACE>`.

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Terminal input

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get chi
kubectl -n <NAMESPACE> logs deployment/<DEPLOYMENT_NAME>
kubectl -n <NAMESPACE> logs -l clickhouse.altinity.com/chi=<RELEASE_NAME>
```


Desinstalação

Terminal input

```
helm -n <NAMESPACE> uninstall <RELEASE_NAME>
```

Cleanup pré-delete

Quando `tdp-clickhouse.cleanup.enabled: true`, o chart habilita um Job de cleanup executado antes da remoção do release. Esse Job remove o `ClickHouseInstallation` e executa limpeza de órfãos para `StatefulSets`, Pods e Services relacionados ao CHI.

Os PVCs são preservados por padrão. O cleanup também não remove diretamente o Deployment do Altinity Operator.

© Copyright Tecnisys 2026. All rights reserved.

ClickHouse – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 25.8.11.66

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Esta página concentra os aspectos de **segurança e controle de acesso** do `tdp-clickhouse`: senhas, usuário `default`, usuário TDP, perfis adicionais e autenticação LDAP.

Para configurações de **integração com S3/MinIO** e políticas de armazenamento, consulte Integrações — ClickHouse.

Pré-requisitos

- Servidor LDAP acessível a partir do cluster Kubernetes, quando LDAP for usado.
- Bloco `tdp-clickhouse.clickhouse.extraConfig` preparado com `<ldap_servers>` e `<user_directories>`, quando LDAP for habilitado.
- Planejamento das roles e grants do ClickHouse para usuários autenticados via LDAP.

Gestão de senhas locais

Usuário default

Configure `tdp-clickhouse.clickhouse.defaultUser.password` apenas quando precisar definir a senha do usuário `default`. Se não houver essa necessidade, mantenha o valor vazio.

```
tdp-clickhouse:
  clickhouse:
    defaultUser:
      password: "<PASSWORD>"
```

© Copyright Tecnisys 2026. All rights reserved.

Usuário TDP (opcional)

Quando `tdpUser.enabled: true`, o chart cria um Secret (`tdp-clickhouse-tdp-user`) e executa um Job pós-instalação/atualização para aplicar grants. O Job aguarda os endpoints do serviço ClickHouse e executa os comandos em um pod backend disponível.

```
tdp-clickhouse:
  clickhouse:
    tdpUser:
      enabled: true
      password: "<PASSWORD>"
```

Perfis e usuários adicionais (`extraUsers`)

Use `tdp-clickhouse.clickhouse.extraUsers` para configuração XML adicional de usuários e perfis do ClickHouse. Nesta página, o campo aparece no contexto de controle de acesso de rede, conforme o exemplo a seguir.

Controle de acesso de rede

Por padrão, o ClickHouse aceita conexões apenas de `127.0.0.1/::1`. Para permitir que outras redes (como o Ingress Controller ou outros pods) acessem o ClickHouse, configure o bloco `<networks>` em `extraUsers`:

```
tdp-clickhouse:
  clickhouse:
    extraUsers: |
      <yandex>
        <profiles>
          <default>

<query_cache_system_table_handling>ignore</query_cache_system_table_handling>
          </default>
        </profiles>
        <users>
          <default>
            <networks>
              <ip>::/0</ip>
            </networks>
          </default>
```

© Copyright Tecnisys 2026. All rights reserved.

```
        </users>
      </yandex>
```

LDAP

O chart suporta autenticação LDAP via `extraConfig` (XML nativo do ClickHouse). Por padrão, o LDAP fica **desabilitado**.

Configuração via `extraConfig`

Ative o LDAP descomentando e preenchendo o bloco `extraConfig` (e, opcionalmente, `ldapRoleInit`) em um arquivo de valores separado:

```
tdp-clickhouse:
  clickhouse:
    extraConfig: |
      <yandex>
        <user_directories>
          <ldap>
            <server>tdp_ldap</server>
            <roles>
              <tdp_ldap_role />
            </roles>
          </ldap>
          <local_directory>
            <path>/var/lib/clickhouse/access/</path>
          </local_directory>
          <users_xml>
            <path>users.xml</path>
          </users_xml>
        </user_directories>
        <ldap_servers>
          <tdp_ldap>
            <host>ldap.example.com</host>
            <port>389</port>
            <auth_dn_prefix>uid=</auth_dn_prefix>

<auth_dn_suffix>,cn=users,cn=accounts,dc=example,dc=com</auth_dn_suffix>
            <base_dn>cn=users,cn=accounts,dc=example,dc=com</base_dn>
            <user_filter>(uid={user})</user_filter>
            <enable_tls>0</enable_tls>
          </tdp_ldap>
```

© Copyright Tecnisys 2026. All rights reserved.

```
    </ldap_servers>
  </yandex>
```

Ajuste `host`, `port`, `auth_dn_prefix`, `auth_dn_suffix`, `base_dn` e `user_filter` ao seu ambiente. O método `auth_dn_prefix`/`auth_dn_suffix` constrói o DN de bind diretamente do UID do usuário, sem necessidade de conta de serviço.

> **ATENÇÃO AO TLS DO LDAP**
>
> No exemplo acima, `enable_tls: 0` representa apenas um ponto de partida. Se o seu servidor LDAP suportar TLS, prefira ativá-lo em produção e trate certificados conforme a política da sua organização.

Criação automática da role LDAP (`ldapRoleInit`)

O chart pode criar a role LDAP automaticamente quando `ldapRoleInit.enabled: true`. Bases de dados e GRANTs continuam a ser geridos manualmente após o deploy:

```
-- Opcional se ldapRoleInit.enabled=true
CREATE ROLE IF NOT EXISTS tdp_ldap_role;

-- Grants opcionais: aplique apenas o acesso necessário
-- GRANT ALL ON tdp.* TO tdp_ldap_role;
-- GRANT SELECT ON default.* TO tdp_ldap_role;
-- GRANT SELECT ON system.* TO tdp_ldap_role;
```

Parâmetros LDAP

Parâmetro	Descrição
`tdp-clickhouse.clickhouse.extraConfig`	XML com `<ldap_servers>` e `<user_directories>`
`tdp-clickhouse.clickhouse.ldapRoleInit.enabled`	Cria a role LDAP automaticamente no pós-install

© Copyright Tecnisys 2026. All rights reserved.



ARQUIVO DE VALORES SEPARADO

Coloque blocos LDAP num arquivo separado, por exemplo `values-ldap.yaml`, para evitar expor configurações sensíveis no repositório principal.

© Copyright Tecnisys 2026. All rights reserved.

ClickHouse – Integração

CHART

Version 3.0.1 | Type application | AppVersion 25.8.11.66

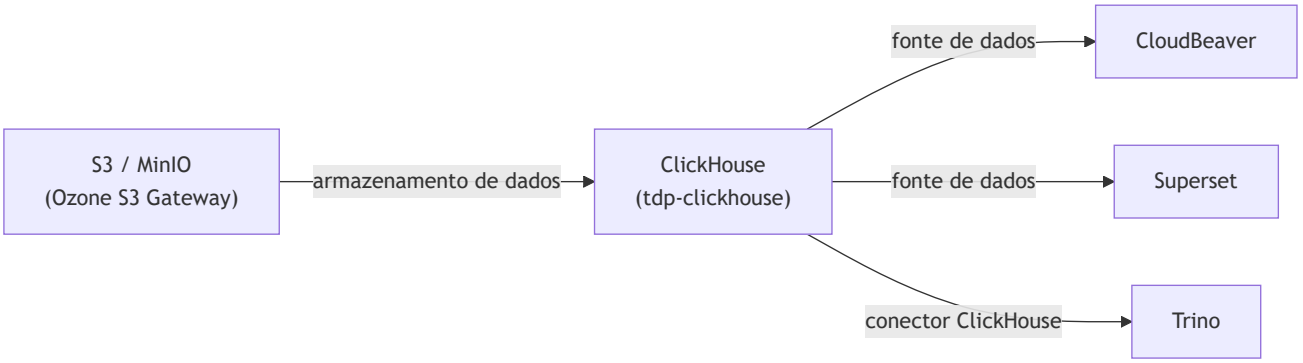
COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Esta página concentra as integrações do `tdp-clickhouse` com **S3/MinIO** e com outras ferramentas do ecossistema TDP.

Aspectos de **senhas, LDAP e controle de acesso** ficam em Segurança — ClickHouse.

Visão geral das integrações



S3 / MinIO

O chart suporta dois níveis diferentes de integração com S3:

1. **Integração do chart com o bucket** via `TDP-Settings.s3Connection`
2. **Política de armazenamento do próprio ClickHouse** via `tdp-clickhouse.clickhouse.extraConfig`

Esses dois níveis são complementares: o primeiro ajuda a preparar o bucket; o segundo ensina o ClickHouse a usar S3 como disco de armazenamento.

Bucket via `TDP-Settings.s3Connection`

© Copyright Tecnisys 2026. All rights reserved.



Quando `s3Connection.enabled: true`, o chart executa um Job que:

- tenta criar o bucket, se ainda não existir;
- usa as credenciais configuradas durante a execução do Job de preparação do bucket;
- cria um Secret (`tdp-clickhouse-s3-connection`) com os detalhes da conexão.

```
TDP-Settings:
  s3Connection:
    enabled: true
    bucket: "<S3_BUCKET>"
    accessKey: "<S3_ACCESS_KEY>"
    secretKey: "<S3_SECRET_KEY>"
    uri: "https://<S3_ENDPOINT>"
    selfSigned: false
```

> **CREDENCIAIS**
>
> Armazene `accessKey` e `secretKey` em um arquivo de valores separado, fora do Git, ou em um Secret Kubernetes já existente. Nunca os inclua diretamente no repositório.

Política de armazenamento S3 no ClickHouse

Se quiser que o ClickHouse armazene dados em S3/MinIO, adicione a configuração XML em `tdp-clickhouse.clickhouse.extraConfig`.

```
tdp-clickhouse:
  clickhouse:
    extraConfig: |
      <yandex>
        <storage_configuration>
          <disks>
            <default>
              <keep_free_space_bytes>1073741824</keep_free_space_bytes>
            </default>
            <s3>
              <type>s3</type>
              <endpoint>https://<S3_ENDPOINT>/<S3_BUCKET>/</endpoint>
              <access_key_id><S3_ACCESS_KEY></access_key_id>
```

© Copyright Tecnisys 2026. All rights reserved.

```
              <secret_access_key><S3_SECRET_KEY></secret_access_key>
              <use_https>1</use_https>

<skip_hostname_verification_for_https>1</skip_hostname_verification_for_https>
              <bucket><S3_BUCKET></bucket>
              <path>data/</path>
              <use_environment_credentials>false</use_environment_credentials>
              <path_style>true</path_style>
            </s3>
          </disks>
          <policies>
            <default>
              <volumes>
                <main>
                  <disk>default</disk>
                </main>
              </volumes>
            </default>
            <s3>
              <volumes>
                <main>
                  <disk>s3</disk>
                </main>
              </volumes>
            </s3>
          </policies>
        </storage_configuration>
        <openSSL>
          <client>
            <loadDefaultCAFile>false</loadDefaultCAFile>
            <verificationMode>none</verificationMode>
            <invalidCertificateHandler>
              <name>AcceptCertificateHandler</name>
            </invalidCertificateHandler>
          </client>
        </openSSL>
      </yandex>
```

Depois de aplicar essa política, é possível criar tabelas usando `storage_policy = 's3'`:

```
CREATE TABLE minha_tabela (id UInt64, valor String)
ENGINE = MergeTree()
```

© Copyright Tecnisys 2026. All rights reserved.

```
ORDER BY id
SETTINGS storage_policy = 's3';
```

> **TLS E CERTIFICADOS**
>
> No exemplo acima, `verificationMode: none` e `AcceptCertificateHandler` devem ser tratados como exemplo para ambientes controlados. Em produção, prefira validação TLS adequada sempre que o endpoint S3 suportar certificados confiáveis.

Parâmetros S3 principais

Parâmetro	Descrição	Padrão
`TDP-Settings.s3Connection.enabled`	Ativar integração S3	`false`
`TDP-Settings.s3Connection.bucket`	Nome do bucket	—
`TDP-Settings.s3Connection.accessKey`	Chave de acesso	—
`TDP-Settings.s3Connection.secretKey`	Chave secreta	—
`TDP-Settings.s3Connection.uri`	Endpoint S3	—
`TDP-Settings.s3Connection.selfSigned`	Aceitar certificado autoassinado	`false`

Consistência entre Job S3 e `extraConfig`

Se você usar os dois mecanismos ao mesmo tempo, mantenha coerentes os seguintes valores:

© Copyright Tecnisys 2026. All rights reserved.

- bucket
- endpoint
- access key
- secret key
- política de TLS/certificados

Isso evita situações em que o bucket é criado com um endpoint, mas o ClickHouse tenta escrever noutro endpoint diferente.

Outras integrações do ecossistema

CloudBeaver

O CloudBeaver pode apontar para o ClickHouse como datasource. A configuração é feita no chart `tdp-cloudbeaver` — consulte Integrações — CloudBeaver.

Superset

O Superset pode consultar o ClickHouse via driver SQLAlchemy. A configuração é feita no chart `tdp-superset` — consulte Integrações — Superset.

Trino

O Trino pode consumir dados do ClickHouse via conector próprio. A configuração é feita no chart `tdp-trino` — consulte Integrações — Trino.

Combinando arquivos de valores

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> \
  -f meu-values.yaml \
  -f values-integracao.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

CloudBeaver

CHART

Version 3.0.1 | Type application | AppVersion 25.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | External DB suportado | Ingress suportado | Gateway API suportado

O que é o CloudBeaver?

O CloudBeaver é um gerenciador de bancos de dados em nuvem que oferece uma interface web completa para administração de bancos de dados. Suporta múltiplos tipos de bancos de dados e oferece recursos como edição de SQL, navegação de dados e gerenciamento de esquemas. É a versão web do DBeaver — acessível pelo navegador, sem necessidade de instalar nada no computador do usuário.

No contexto do TDP, o CloudBeaver é a interface centralizada para explorar e consultar dados no **ClickHouse**, **Trino** e **PostgreSQL** diretamente pelo browser, permitindo que desenvolvedores e analistas acessem dados sem instalação local de ferramentas.

O chart `tdp-cloudbeaver` empacota o CloudBeaver 25.3.0 para Kubernetes.

> **PARA SABER MAIS**
>
> Consulte CloudBeaver — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Por que usar o CloudBeaver no TDP?

© Copyright Tecnisys 2026. All rights reserved.



No TDP Kubernetes, os bancos de dados ficam dentro do cluster e não são acessíveis diretamente da sua máquina. O CloudBeaver resolve esse problema: roda dentro do cluster, tem acesso a todos os serviços de banco de dados, e você acessa tudo via um único endereço web.

Componentes implantados

Componente	Descrição
CloudBeaver Server	Servidor web que gerencia conexões e sessões
Workspace	Volume persistente com configurações e workspaces dos usuários
Integração com PostgreSQL externo (opcional)	Banco usado para metadados do CloudBeaver quando `tdp-cloudbeaver.database.enabled: true`

Estrutura de valores (Helm)

O chart `tdp-cloudbeaver` distribui a configuração em quatro blocos principais:

- **`tdp-cloudbeaver:`** — configuração do servidor CloudBeaver: imagem, serviço, persistência, credenciais admin, datasources, LDAP, banco de metadados externo.
- **`gatewayApi:`** — configuração de Gateway API (HTTPRoute e Gateway para a interface web).
- **`TDP-Settings:`** — chave de controle da exposição HTTP/HTTPS: escolhe entre Ingress e Gateway API.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: false

tdp-cloudbeaver:
```

© Copyright Tecnisys 2026. All rights reserved.

```
  persistence:
    size: 5Gi
```

Visão geral

Propriedade	Valor
Chart	`tdp-cloudbeaver`
Versão do CloudBeaver	25.3.0
Versão do chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver`
Porta padrão	`8978`
Persistência padrão	`5Gi`

Páginas relacionadas

- Integrações — CloudBeaver: ClickHouse, Trino, S3/Ozone e configuração automática (`initialData`).
- Segurança — CloudBeaver: autenticação LDAP.
- Exposição externa: exposição HTTP/HTTPS por Ingress ou Gateway API.

Pré-requisitos

Antes de instalar o CloudBeaver, certifique-se de que sua infraestrutura atende aos seguintes requisitos:

- **Kubernetes 1.32+**, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- **Helm 3.2.0+**

Instalação

Instalação

© Copyright Tecnisys 2026. All rights reserved.

Instalação mínima via registry OCI:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  --version <CHART_VERSION> \
  -n <NAMESPACE> \
  --create-namespace \
  -f <VALUES_FILE>
```

EXEMPLO

Exemplo

```
helm upgrade --install cloudbeaver \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  --version 3.0.1 \
  -n tdp-project \
  --create-namespace \
  -f values-cloudbeaver.yaml
```

OpenShift

O chart é compatível com OpenShift. Antes da instalação, confirme a política de SCC adotada no cluster e alinhe os ajustes de Security Context com o padrão operacional do ambiente.

Para contextualizar OpenShift e Rancher, use o tópico **Compatibilidade do orquestrador e ajustes de segurança** na página Configuração Geral, aba **Decisões de ambiente que afetam a configuração**.

Verificação pós-instalação

Após a instalação, verifique se o CloudBeaver iniciou corretamente:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
# Estado dos pods
kubectl get pods -n <NAMESPACE>

# Serviço
kubectl get svc -n <NAMESPACE>

# PVC
kubectl get pvc -n <NAMESPACE>

# Logs do container
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver
```

Parâmetros principais

Parâmetros principais

Parâmetros do servidor

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.image.registry`	Registry da imagem de container	`docker.io`
`tdp-cloudbeaver.image.repository`	Repositório da imagem	`dbeaver/cloudbeaver`
`tdp-cloudbeaver.image.tag`	Tag da imagem	`25.3.0`
`tdp-cloudbeaver.service.type`	Tipo de Service	`ClusterIP`
`tdp-cloudbeaver.service.nodePorts.http`	NodePort HTTP (só quando `service.type: NodePort`)	`""`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.persistence.enabled`	Habilitar persistência	`true`
`tdp-cloudbeaver.persistence.size`	Tamanho do volume persistente	`5Gi`
`tdp-cloudbeaver.persistence.storageClass`	StorageClass (vazio = padrão do cluster)	`""`
`tdp-cloudbeaver.persistence.existingClaim`	PVC existente (vazio = cria novo)	`""`
`tdp-cloudbeaver.resources.limits.cpu`	Limite de CPU	`1`
`tdp-cloudbeaver.resources.limits.memory`	Limite de memória	`2Gi`
`tdp-cloudbeaver.resources.requests.cpu`	Requisição de CPU	`500m`
`tdp-cloudbeaver.resources.requests.memory`	Requisição de memória	`1Gi`
`tdp-cloudbeaver.ingress.ingressClassName`	IngressClass	`""`
`tdp-cloudbeaver.ingress.hostname`	Hostname para Ingress	`cloudbeaver.tdp.local`
`tdp-cloudbeaver.ingress.path`	Path do Ingress	`/`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.ingress.pathType`	Tipo de path	`ImplementationSpecific`
`tdp-cloudbeaver.ingress.annotations`	Anotações do Ingress (ex.: timeouts nginx)	`{}`
`tdp-cloudbeaver.ingress.tls`	Habilitar TLS no Ingress	`false`
`TDP-Settings.gateway.ingress.enabled`	Criar recurso Ingress	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Criar recursos Gateway API	`false`

Parâmetros de configuração automática (initialData)

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.initialData.enabled`	Habilitar configuração automática	`false`
`tdp-cloudbeaver.initialData.name`	Nome do ConfigMap para initial-data	`""`
`tdp-cloudbeaver.initialData.adminName`	Usuário administrador (não pode ser `admin`)	`""`
`tdp-cloudbeaver.initialData.adminPassword`	Senha do administrador (⚠ use Secret em produção)	`""`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.adminSecret.enabled`	Criar Secret para credenciais do admin	`false`
`tdp-cloudbeaver.adminSecret.name`	Nome do Secret	`""`
`tdp-cloudbeaver.adminSecret.data`	Dados do Secret (pares chave-valor)	`{}`

Parâmetros de banco de dados externo

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.database.enabled`	Habilitar banco de dados PostgreSQL externo	`false`
`tdp-cloudbeaver.database.type`	Tipo do banco de dados	`postgresql`
`tdp-cloudbeaver.database.host`	Host do PostgreSQL	`""`
`tdp-cloudbeaver.database.port`	Porta do PostgreSQL	`5432`
`tdp-cloudbeaver.database.databaseName`	Nome do banco de dados a criar	`""`
`tdp-cloudbeaver.database.schema`	Schema do banco de dados	`public`
`tdp-cloudbeaver.database.user`	Usuário do banco de dados do CloudBeaver	`""`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.database.driver`	Driver do banco de dados do CloudBeaver	`postgres-jdbc`
`tdp-cloudbeaver.database.passwordSecret.name`	Nome do Secret com a senha do banco	`""`
`tdp-cloudbeaver.database.passwordSecret.key`	Chave do Secret com a senha do banco	`password`
`tdp-cloudbeaver.database.postgresSecret.name`	Nome do Secret com a senha do admin do postgres	`""`
`tdp-cloudbeaver.database.postgresSecret.key`	Chave do Secret com a senha do admin do postgres	`postgres-password`
`tdp-cloudbeaver.database.secret.enabled`	Criar Secret de senha do banco via Helm	`false`

Detalhes de configuração

Como funciona o PostgreSQL externo

Quando `tdp-cloudbeaver.database.enabled: true`, o chart configura o CloudBeaver para usar um PostgreSQL externo como banco de metadados em vez do banco embutido. Esse modo é indicado para produção porque facilita backup, escala e gestão operacional.

O fluxo executado pelo chart é:

1. criar um Job de pré-instalação para conectar ao PostgreSQL e preparar o banco dedicado do CloudBeaver;
2. criar o usuário dedicado do CloudBeaver com as permissões necessárias;

© Copyright Tecnisys 2026. All rights reserved.



3. configurar o container do CloudBeaver por variáveis de ambiente `CLOUDBEAVER_DB_*`;
4. iniciar o CloudBeaver já apontando para o banco externo.

As credenciais envolvem dois Secrets distintos:

Secret	Uso
`tdp-cloudbeaver.database.passwordSecret`	Senha do usuário de banco usado pelo CloudBeaver
`tdp-cloudbeaver.database.postgresSecret`	Senha do usuário administrador do PostgreSQL usada pelo Job de criação

Variáveis usadas pelo CloudBeaver para conexão:

Variável	Uso
`CLOUDBEAVER_DB_DRIVER`	Driver JDBC, como `postgres-jdbc`
`CLOUDBEAVER_DB_URL`	URL JDBC do PostgreSQL
`CLOUDBEAVER_DB_SCHEMA`	Schema de banco usado pelo CloudBeaver
`CLOUDBEAVER_DB_USER`	Usuário de banco do CloudBeaver
`CLOUDBEAVER_DB_PASSWORD`	Senha lida por `secretKeyRef`

Para validar a criação do banco, verifique o Job e os logs:

Terminal input

```
kubectl get jobs -n <NAMESPACE>
kubectl describe job <RELEASE_NAME>-create-db -n <NAMESPACE>
kubectl logs -n <NAMESPACE> -l job-name=<RELEASE_NAME>-create-db
```

© Copyright Tecnisys 2026. All rights reserved.

Se o CloudBeaver não conectar ao banco externo, confirme as variáveis de ambiente no Pod e procure erros de banco nos logs:

Terminal input

```
kubectl get pod -n <NAMESPACE> -o yaml | grep -A 20 "env:"
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep -i database
```

Para a lista completa de parâmetros, consulte `helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver`.

Configuração automática (initialData)

O chart suporta `initialData.enabled: true` para provisionar o CloudBeaver automaticamente, com administrador e configurações iniciais pré-definidas, sem necessidade de assistente de configuração.

> **PRÉ-REQUISITO: WORKSPACE LIMPO**
>
> A configuração automática só funciona na **primeira inicialização com workspace vazio**. Se o CloudBeaver já foi configurado, remova o PVC do workspace antes de reinstalar.

Para habilitar a configuração automática, defina `initialData.enabled: true` e crie ou referencie o Secret com a senha do administrador antes da instalação:

```yaml
tdp-cloudbeaver:
  initialData:
    enabled: true
    name: tdp-cloudbeaver-initial-data
    adminName: cbadmin  # Não pode ser "admin"

  adminSecret:
    enabled: true
    name: tdp-cloudbeaver-admin-secret
    data:
      password: "<PASSWORD>"  # Nunca commite a senha em repositórios Git
```

© Copyright Tecnisys 2026. All rights reserved.

Para detalhes de initialData, validação e resolução de problemas, consulte Integrações — CloudBeaver.

Integrações (ClickHouse, Trino, S3)

Configure integrações com fontes de dados e armazenamento no arquivo de valores. Consulte Integrações — CloudBeaver para instruções detalhadas, exemplos e resolução de problemas.

Ambiente de desenvolvimento (recursos menores)

O trecho abaixo reduz CPU, memória e tamanho de PVC para **clusters de desenvolvimento ou laboratório** com poucos usuários simultâneos.

```
tdp-cloudbeaver:
  service:
    type: ClusterIP
  resources:
    limits:
      cpu: "500m"
      memory: "1Gi"
    requests:
      cpu: "250m"
      memory: "512Mi"
  persistence:
    size: 2Gi
```

PVC existente

Se quiser usar um PVC existente em vez de criar um novo:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE> \
  --set tdp-cloudbeaver.persistence.existingClaim=<PVC_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.



> **NOTE**
>
> O PVC informado em `existingClaim` deve existir previamente no namespace de instalação antes de executar este comando.

Acesso & Segurança

Acesso

Com `service.type: ClusterIP` (padrão), o acesso é feito por **port-forward** para validação local ou por **Ingress / Gateway API** para acesso externo com hostname estável.

Port-forward

Para acesso local:

Terminal input

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME> 8978:8978
```

Em seguida, acesse o CloudBeaver em `http://localhost:8978`.

NodePort (opcional)

Se o Service for configurado com `service.type: NodePort`, acesse pelo IP de um nó e a NodePort alocada:

Terminal input

```
kubectl -n <NAMESPACE> get svc <RELEASE_NAME>
kubectl -n <NAMESPACE> get nodes -o wide
```

Em seguida, acesse o CloudBeaver em `http://<NODE_IP>:<NODE_PORT>`.

Exposição externa

© Copyright Tecnisys 2026. All rights reserved.



Para publicar a interface web com hostname estável, use Ingress ou Gateway API. Consulte Exposição externa — CloudBeaver para os exemplos completos de Ingress, Gateway API, TLS e resolução de problemas.

Segurança (LDAP)

Para autenticação LDAP, consulte Segurança — CloudBeaver.

Solução de problemas

Solução de problemas

Pod não iniciando

Verifique os logs do pod:

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver
```

Configuração automática não funcionando (assistente de configuração ainda aparece)

1. **Verifique se o ConfigMap está montado**:

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- ls -la /opt/cloudbeaver/conf/
kubectl exec -n <NAMESPACE> <POD_NAME> -- cat /opt/cloudbeaver/conf/initial-data.conf
```

2. **Verifique se o workspace está limpo**:

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- ls -la /opt/cloudbeaver/workspace/.data/
```

   Se você encontrar `.cloudbeaver.runtime.conf`, o workspace não está limpo. Exclua o PVC e reinstale:

© Copyright Tecnisys 2026. All rights reserved.

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
kubectl delete pvc -n <NAMESPACE> --all
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

3. **Verifique o conteúdo do ConfigMap**:

```
kubectl get configmap tdp-cloudbeaver-initial-data -n <NAMESPACE> -o yaml
```

4. **Verifique os logs do CloudBeaver**:

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep
-i config
```

Problemas com volume persistente

Verifique o status do PVC:

Terminal input

```
kubectl get pvc -n <NAMESPACE>
kubectl describe pvc -n <NAMESPACE>
```

Serviço não acessível

Verifique os endpoints do serviço:

Terminal input

```
kubectl get endpoints -n <NAMESPACE>
kubectl describe svc <SERVICE_NAME> -n <NAMESPACE>
```

Falha no job de criação do banco de dados

1. **Verifique o status do job**:

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl get jobs -n <NAMESPACE>
kubectl describe job <RELEASE_NAME>-create-db -n <NAMESPACE>
```

2. **Veja os logs do job**:

```
kubectl logs -n <NAMESPACE> -l job-name=<RELEASE_NAME>-create-db
```

3. **Problemas comuns**:
   - **Secret não encontrado**: Verifique se `database.postgresSecret` e `database.passwordSecret` existem
   - **Conexão recusada**: Confirme que host e porta do PostgreSQL estão corretos
   - **Falha de autenticação**: Verifique a senha do admin do PostgreSQL no secret
   - **Permissão negada**: Confirme que o usuário postgres tem permissão para criar bancos de dados
4. **Limpeza manual** (se necessário):

```
kubectl delete job <RELEASE_NAME>-create-db -n <NAMESPACE>

helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

CloudBeaver não conecta ao banco de dados externo

1. **Verifique as variáveis de ambiente**:

```
kubectl get pod -n <NAMESPACE> -o yaml | grep -A 20 "env:"
```

2. **Verifique os logs do CloudBeaver em busca de erros de banco de dados**:

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep
-i database
```

3. **Teste a conexão com o banco de dados a partir do pod**:

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl exec -it -n <NAMESPACE> <POD_NAME> -- /bin/bash

# Instale o cliente psql (se necessário)
apt-get update && apt-get install -y postgresql-client

# Teste a conexão
psql -h <DATABASE_HOST> -U <DATABASE_USER> -d <DATABASE_NAME>
```

Desinstalação

Desinstalação

Remover o release Helm

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

EXEMPLO

Exemplo

```
helm uninstall cloudbeaver -n tdp-project
```

Este comando remove os recursos do release (pods, services, ConfigMaps, Secrets). Os PVCs (volumes) são mantidos por padrão para preservar dados e configurações.

Para remover também os PVCs associados ao chart (operação destrutiva e irreversível):

Terminal input

```
kubectl delete pvc -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver
```

Excluir o namespace

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl delete namespace <NAMESPACE>
```

> **WARNING**
>
> Isso irá excluir todos os recursos do namespace, incluindo volumes persistentes que não tiverem política de retenção.

© Copyright Tecnisys 2026. All rights reserved.

CloudBeaver – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 25.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado

Configure autenticação LDAP para o CloudBeaver Community, permitindo que usuários se autentiquem contra seu servidor LDAP corporativo em vez de usar contas locais.

Siga as práticas de **não commitar senhas** em arquivos versionados e usar **Kubernetes Secrets** ou operadores de gestão de secrets.

Pré-requisitos

- Servidor LDAP alcançável a partir do cluster (ex.: `ldap.empresa.local:389`)
- DN de bind com permissão para buscar usuários no diretório
- Credenciais do bind (usuário e senha)
- Base DN onde os usuários estão localizados
- Filtro LDAP para localizar usuários por ID

Credenciais e Secrets

Terminal input

```
kubectl -n <NAMESPACE> create secret generic tdp-cloudbeaver-ldap-bind \
  --from-literal=password='<LDAP_BIND_PASSWORD>'
```

Este Secret armazena a senha de bind de forma segura, referenciada pelos valores do chart.

© Copyright Tecnisys 2026. All rights reserved.

Configuração LDAP

A configuração LDAP usa os blocos `ldap.server`, `ldap.bind`, `ldap.userSearch` e `ldap.extraEnvVars` para injetar `CLOUDBEAVER_AUTH_LDAP_BIND_PASSWORD` via `secretKeyRef`.

O chart gera automaticamente o arquivo `cloudbeaver.conf` com o bloco `authConfigurations` a partir desses campos — não é necessário fornecer JSON manualmente.

```
tdp-cloudbeaver:
  ldap:
    enabled: true
    server:
      host: "<LDAP_HOST>"
      port: 389
      useSSL: false
      useTLS: false
      insecureSkipVerify: true
    bind:
      dn: "<LDAP_BIND_DN>"
      passwordSecretName: "tdp-cloudbeaver-ldap-bind"
      passwordSecretKey: "password"
    userSearch:
      baseDN: "<LDAP_USER_BASE_DN>"
      filter: "(objectClass=person)"
      userIdAttribute: "uid"
    extraEnvVars: |
      - name: CLOUDBEAVER_AUTH_LDAP_BIND_PASSWORD
        valueFrom:
          secretKeyRef:
            name: tdp-cloudbeaver-ldap-bind
            key: password
```

> **CONFIGURAÇÃO GERADA AUTOMATICAMENTE PELO CHART**
>
> O chart gera o arquivo `cloudbeaver.conf` com o bloco `authConfigurations` automaticamente, a partir dos campos `ldap.server.*`, `ldap.bind.*` e `ldap.userSearch.*`.
> Não é necessário definir `settings` manualmente nos valores.

© Copyright Tecnisys 2026. All rights reserved.



> O campo `ldap.extraEnvVars` injeta a senha de bind em tempo de execução via `secretKeyRef`.

Placeholders a substituir:

- `<LDAP_HOST>`: hostname ou IP do servidor LDAP
- `<LDAP_BIND_DN>`: DN completo do usuário de bind (ex.: `cn=admin,dc=empresa,dc=local`)
- `<LDAP_USER_BASE_DN>`: Base DN onde os usuários estão (ex.: `ou=pessoas,dc=empresa,dc=local`)

Instalação ou atualização

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

Parâmetros

Área	Uso	Exemplo
`ldap.server.*`	Host, porta, SSL/TLS	`host: "ldap.empresa.local"`
`ldap.bind.*`	DN de bind + nome do Secret com senha	`dn: "cn=admin,dc=empresa,dc=local"`
`ldap.userSearch.*`	Base DN, filtro, atributo de login	`baseDN: "ou=pessoas,dc=empresa,dc=local"`
`ldap.extraEnvVars`	Injeção da senha de bind via `secretKeyRef`	Sempre incluir `CLOUDBEAVER_AUTH_LDAP_BIND_PASSWORD`

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Opção LDAP não aparece no login

Verificação 1: confirmar se LDAP está ativado

```
tdp-cloudbeaver:
  ldap:
    enabled: true
```

Verificação 2: validar a seção `authConfigurations` no ConfigMap gerado

Terminal input

```
kubectl get configmap tdp-cloudbeaver-auto-config -n <NAMESPACE> -o yaml | \
  grep -A 20 "authConfigurations"
```

Verifique se o bloco contém o provider `"ldap"` e se o host e DN estão corretos.

Verificação 3: verificar logs do pod

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep -i
ldap
```

Se não houver mensagens sobre LDAP, pode indicar que a configuração não foi reconhecida.

Falha de autenticação LDAP

Verificação 1: testar conectividade ao servidor LDAP

Terminal input

```
kubectl exec -it -n <NAMESPACE> <POD_NAME> -- nc -zv <LDAP_HOST> 389
```

© Copyright Tecnisys 2026. All rights reserved.

Se a conexão for recusada, o LDAP não está alcançável.

Verificação 2: verificar credenciais de bind

Terminal input

```
# Executar dentro do pod ou de um pod utilitário com ldap-utils
ldapwhoami -x -H ldap://<LDAP_HOST>:389 \
  -D "<LDAP_BIND_DN>" \
  -w '<LDAP_BIND_PASSWORD>'
```

Se falhar, a senha de bind ou o DN estão incorretos.

Verificação 3: validar filtro e base DN

Terminal input

```
# Buscar usuários na base DN com o filtro configurado
ldapsearch -x -H ldap://<LDAP_HOST>:389 \
  -D "<LDAP_BIND_DN>" \
  -w '<LDAP_BIND_PASSWORD>' \
  -b "<LDAP_USER_BASE_DN>" \
  "<LDAP_FILTER>"
```

Se não encontrar usuários, revise o filtro ou a base DN.

Verificação 4: verificar atributo de ID do usuário

Terminal input

```
# Buscar um usuário específico e ver quais atributos possui
ldapsearch -x -H ldap://<LDAP_HOST>:389 \
  -D "<LDAP_BIND_DN>" \
  -w '<LDAP_BIND_PASSWORD>' \
  -b "<LDAP_USER_BASE_DN>" \
  "uid=<USERNAME>"
```

© Copyright Tecnisys 2026. All rights reserved.

Confirme que `userIdAttribute` corresponde a um atributo existente (ex.: `uid`, `mail`, `sAMAccountName`).

Atributo de usuário não encontrado

Sintoma: usuário existe no LDAP mas CloudBeaver não consegue localizá-lo.

Verificação: inspecionar atributos do usuário no LDAP:

Terminal input

```
ldapsearch -x -H ldap://<LDAP_HOST>:389 \
  -D "<LDAP_BIND_DN>" -W \
  -b "<LDAP_USER_BASE_DN>" \
  "(uid=<USERNAME>)" uid givenName sn mail sAMAccountName
```

Se o usuário não tiver o atributo definido em `userIdAttribute` (padrão `uid`), atualize o valor — por exemplo, para `sAMAccountName` em ambientes Active Directory.

Logs detalhados do CloudBeaver

Para ver logs completos de autenticação:

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver --tail=100
| grep -i "auth\|ldap\|bind"
```

Procure por:

- Mensagens de conexão ao LDAP
- Tentativas de bind
- Erros de busca de usuários
- Mensagens de sucesso/falha de autenticação

Fluxo de autenticação LDAP

1. O usuário seleciona **"LDAP"** na tela de login do CloudBeaver.

© Copyright Tecnisys 2026. All rights reserved.

2. O CloudBeaver realiza um **bind** ao servidor LDAP usando as credenciais configuradas em `ldap.bind`.
3. Busca o usuário com o filtro definido em `ldap.userSearch.filter` e o atributo `userIdAttribute`.
4. Autentica o usuário com a senha informada no login.
5. Cria ou atualiza o perfil do usuário no CloudBeaver com os atributos LDAP.

Considerações de segurança

- **Use LDAPS em produção**: habilite `ldap.server.useSSL: true` para criptografar o tráfego LDAP.
- **Usuário de bind dedicado**: use um usuário de bind com permissões mínimas (somente leitura no diretório).
- **Segurança de rede**: o servidor LDAP deve ser acessível apenas de redes autorizadas.
- **Gestão de senhas**: nunca commite senhas em Git; use Kubernetes Secrets com Sealed Secrets, External Secrets Operator ou HashiCorp Vault em produção.
- `insecureSkipVerify`: defina como `false` em produção para validar o certificado TLS do servidor LDAP.

LDAP vs autenticação local

O CloudBeaver suporta autenticação local e LDAP simultaneamente:

Tipo	Usuários	Quando usar
Local	Usuários criados diretamente no CloudBeaver (ex.: `admin`)	Sempre disponível; necessário para administração inicial
LDAP	Usuários do diretório corporativo	Ambientes com SSO ou autenticação centralizada

> **USUÁRIO ADMIN LOCAL**
>
> O usuário admin local (configurado via `initialData.adminName`) permanece disponível independentemente do LDAP.

© Copyright Tecnisys 2026. All rights reserved.

Mantenha-o como contingência para acesso administrativo caso o LDAP fique indisponível.

© Copyright Tecnisys 2026. All rights reserved.

CloudBeaver – Integração

CHART

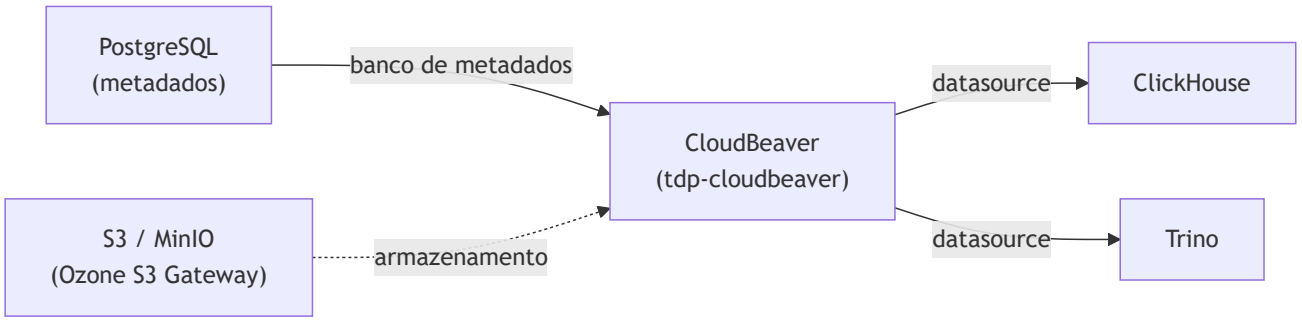
Version 3.0.1 | Type application | AppVersion 25.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Esta página detalha como configurar integrações do CloudBeaver com **ClickHouse**, **Trino**, **S3/Ozone** e **initialData** (provisionamento automático). O banco de metadados (PostgreSQL) é configurado na página de Configuração do CloudBeaver.

Visão geral das integrações



Escopo das integrações

Use cada bloco para uma finalidade diferente:

Integração	Quando usar
ClickHouse	Pré-cadastrar uma conexão direta ao ClickHouse na UI do CloudBeaver
Trino	Pré-cadastrar uma conexão SQL fede-rada pelo Trino

© Copyright Tecnisys 2026. All rights reserved.

Integração	Quando usar
S3/Ozone	Injetar variáveis de ambiente para acesso S3 pela aplicação
initialData	Pular o assistente inicial e criar o administrador na primeira inicialização

ClickHouse

Habilite esta integração quando os usuários precisarem **consultar o ClickHouse nativo** (HTTP 8123) — por exemplo explorar tabelas colunares e executar SQL diretamente no cluster analítico.
Se o fluxo principal da empresa passa só por **Trino** como camada SQL unificada, a integração Trino pode ser suficiente; ClickHouse direto continua útil para diagnóstico, tuning ou acesso a objetos expostos apenas no CH.

Arquitetura

O chart implementa uma configuração segura da fonte de dados com:

1. **Kubernetes Secret**: armazena a senha do ClickHouse
2. **ConfigMap**: contém o modelo `data-sources.json`
3. **InitContainer**: copia a configuração para o *workspace* com as permissões corretas
4. **PVC**: persiste a configuração

```
  CloudBeaver Pod

InitContainer (busybox)
 Mounts ConfigMap (read)
 Copies to PVC (write)
 Sets permissions

CloudBeaver Container
 Reads workspace
 Can modify via UI
 Changes persist

 ConfigMap          Secret
```

© Copyright Tecnisys 2026. All rights reserved.

Opção 1: Secret criado pelo Helm (desenvolvimento)

Para ambientes de desenvolvimento, deixe o Helm criar o Secret:

```
# values-clickhouse.yaml
tdp-cloudbeaver:
  image:
    registry: registry.tecnisys.com.br
    repository: community/images/dbeaver/cloudbeaver
    tag: latest

  dataSources:
    enabled: true
    enabledClickhouse: true
    clickhouseHost: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
    clickhousePort: 8123
    database: "default"
    user: "default"

    # Criar Secret via Helm
    secret:
      enabled: true
      name: <SECRET_NAME>
      data:
        password: "123456"  # ALTERE EM PRODUÇÃO!
```

> **SENHA EM TEXTO CLARO NO VALUES.YAML**
>
> O campo `dataSources.secret.data.password` e `dataSources.clickhouseTdpUser.password` gravam o valor diretamente no `data-sources.json` gerado pelo chart.
> ConfigMaps **não são criptografados** no `etcd` por padrão.
> Use esta abordagem **apenas em desenvolvimento local**.
> Em produção, utilize `dataSources.passwordSecret` para referenciar um Kubernetes Secret criado e gerenciado fora do chart.

Implante:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f meu-values.yaml -f meu-values-clickhouse.yaml
```

Opção 2: Referência a um Secret existente (produção)

Em produção, crie o Secret manualmente e referencie-o:

Passo 1: criar o Secret:

Terminal input

```
kubectl create secret generic <SECRET_NAME> \
  --from-literal=password='SecurePassword123!' \
  -n <NAMESPACE>
```

Passo 2: referenciar no arquivo de valores:

```
tdp-cloudbeaver:
  dataSources:
    enabled: true
    enabledClickhouse: true
    clickhouseHost: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
    clickhousePort: 8123
    database: "default"
    user: "default"

    # Referenciar Secret existente
    passwordSecret:
      name: <SECRET_NAME>
      key: password

    # Desativar Secret gerido pelo Helm
    secret:
      enabled: false
```

Passo 3: implantar:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f meu-values.yaml
```

GESTÃO SEGURA DE SECRETS EM PRODUÇÃO

Para proteger credenciais em repositórios Git e no cluster, considere:

- **Sealed Secrets** — criptografa o Secret no Git; decripta no cluster via `kubeseal`
- **External Secrets Operator (ESO)** — sincroniza segredos de cofres externos (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, etc.)
- **HashiCorp Vault** — cofre centralizado com controle de acesso granular

Nunca commite senhas reais em repositórios Git.

Opção 3: Secret compartilhado entre componentes

Crie um Secret compartilhado usado pelo ClickHouse e pelo CloudBeaver:

Terminal input

```
kubectl create secret generic <SECRET_NAME> \
  --from-literal=password='SharedPassword123!' \
  -n <NAMESPACE>
```

Os dois charts referenciam o mesmo Secret:

ClickHouse:

Terminal input

```
helm upgrade --install <CLICKHOUSE_RELEASE>
oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  -n <NAMESPACE> --create-namespace \
  --set tdp-clickhouse.clickhouse.defaultUser.password="SharedPassword123!"
```

CloudBeaver:

```
tdp-cloudbeaver:
  dataSources:
    enabledClickhouse: true
    passwordSecret:
      name: <SECRET_NAME>
      key: password
```

> **RBAC PARA SECRETS**
>
> O Secret referenciado em `dataSources.passwordSecret.name` deve existir no **mesmo namespace** do release do CloudBeaver.
> O Kubernetes não permite que pods acessem Secrets de outros namespaces sem configuração explícita.
> Se precisar de acesso cross-namespace, utilize uma ferramenta de sincronização como **External Secrets Operator** ou projete o Secret no namespace correto antes da instalação.

Secrets, RBAC e rotação

Para produção, prefira Secrets criados fora do chart e referenciados por `dataSources.passwordSecret`. Essa abordagem evita gravar senhas no `data-sources.json` e permite que a credencial seja gerida por ferramentas como Sealed Secrets, External Secrets Operator, Vault ou provedores de Secret externos.

Se o processo de deploy precisar ler Secrets de forma explícita, limite as permissões por RBAC ao nome do Secret necessário. Evite permissões amplas sobre todos os Secrets do namespace.

Para rotacionar a senha do ClickHouse:

1. atualize o Secret referenciado por `dataSources.passwordSecret`;
2. recrie o Pod do CloudBeaver para que a aplicação leia a credencial atualizada;
3. confira os logs e valide a conexão na UI.

© Copyright Tecnisys 2026. All rights reserved.

Sincronização de senhas

IMPORTANTE: a senha do ClickHouse no CloudBeaver **TEM** de coincidir com a senha real do ClickHouse.

Verificação manual

Terminal input

```
# Verificar senha do ClickHouse no Helm
helm get values <CLICKHOUSE_RELEASE> -n <NAMESPACE> | grep -A2 "defaultUser"

# Verificar ConfigMap do CloudBeaver
kubectl get configmap <CONFIGMAP_NAME> -n <NAMESPACE> -o yaml

# Verificar Secret do CloudBeaver
kubectl get secret <SECRET_NAME> -n <NAMESPACE> \
  -o jsonpath='{.data.password}' | base64 -d && echo
```

Atualizar senhas

Para executar a rotação da senha do ClickHouse:

Terminal input

```
# 1. Atualizar Secret
kubectl create secret generic <SECRET_NAME> \
  --from-literal=password='NewPassword456!' \
  -n <NAMESPACE> \
  --dry-run=client -o yaml | kubectl apply -f -

# 2. Apagar o pod para recriar com o novo Secret
kubectl delete pod -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver

# 3. Verificar logs
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep -i
clickhouse
```

Parâmetros de configuração (ClickHouse)

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.dataSources.enabled`	Habilitar fonte ClickHouse	`false`
`tdp-cloudbeaver.dataSources.name`	Nome do ConfigMap de datasources	`tdp-cloudbeaver-data-sources`
`tdp-cloudbeaver.dataSources.enabledClickhouse`	Ativar ClickHouse	`false`
`tdp-cloudbeaver.dataSources.clickhouseHost`	Host do ClickHouse	`localhost`
`tdp-cloudbeaver.dataSources.clickhousePort`	Porta HTTP do ClickHouse	`8123`
`tdp-cloudbeaver.dataSources.database`	Banco de dados padrão	`default`
`tdp-cloudbeaver.dataSources.user`	Usuário	`default`
`tdp-cloudbeaver.dataSources.passwordSecret.name`	Nome do Secret	`""`
`tdp-cloudbeaver.dataSources.passwordSecret.key`	Chave do Secret	`password`
`tdp-cloudbeaver.dataSources.secret.enabled`	Criar Secret via Helm	`false`
`tdp-cloudbeaver.dataSources.clickhouseTdpUser.enabled`	Criar Secret para o usuário TDP do ClickHouse	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-cloudbeaver.dataSources.clickhouseTdpUser.password`	Senha do usuário TDP (⚠ não recomendado em produção)	`""`

Resolução de problemas (ClickHouse)

A conexão aparece, mas a senha está vazia

Se `passwordSecret` estiver configurado, a senha é deixada vazia de propósito em `data-sources.json`, para não guardar senhas em texto claro em ConfigMaps.

Solução: o CloudBeaver pedirá a senha no primeiro uso.
Informe uma vez e ela ficará armazenada no PVC.

Erro "Read-only file system"

O arquivo `data-sources.json` precisa ser gravável pelo CloudBeaver.

Verificar se o InitContainer executou:

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver \
  -c sync-workspace-config
```

Verificar permissões do arquivo:

Terminal input

```
kubectl exec -n <NAMESPACE> deployment/<DEPLOYMENT_NAME> -- \
  ls -la /opt/cloudbeaver/workspace/GlobalConfiguration/.dbeaver/data-sources.json
```

Esperado: `-rw-rw-r-- 1 ...`

Correção: apague o pod para forçar recriação:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl delete pod -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver
```

Trino

Use o conector Trino quando a equipe deve **consultar fontes já expostas pelo Trino** com a mesma experiência SQL na UI, em linha com o padrão de consumo do TDP descrito na introdução desta página.
O host e a porta devem apontar para o serviço Trino do cluster, não para o ClickHouse, salvo se a sua arquitetura explicitamente combinar outro desenho.

```
tdp-cloudbeaver:
  dataSources:
    enabled: true
    enabledTrino: true
    nameConnector: "Trino TDP"
    trinoHost: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
    trinoPort: 8080
    database: "default"
    user: "admin"
```

S3 compatível (Ozone, MinIO)

Use estas variáveis quando a equipe precisar que o CloudBeaver **acesse arquivos ou exports em armazenamento compatível com S3** (Ozone, MinIO ou outro endpoint S3) a partir da aplicação — cenário distinto do cadastro de ClickHouse/Trino como *data sources* JDBC, embora possa coexistir com eles.

Passo 1: Criar Secret com credenciais S3:

Terminal input

```
kubectl create secret generic <S3_SECRET_NAME> \
  --from-literal=access-key='<S3_ACCESS_KEY>' \
  --from-literal=secret-key='<S3_SECRET_KEY>' \
  -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Passo 2: Adicionar ao arquivo de valores:

```
tdp-cloudbeaver:
  extraEnvVars:
    - name: AWS_ACCESS_KEY_ID
      valueFrom:
        secretKeyRef:
          name: "<S3_SECRET_NAME>"
          key: access-key
    - name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: "<S3_SECRET_NAME>"
          key: secret-key
    - name: AWS_DEFAULT_REGION
      value: "<S3_REGION>"
    - name: S3_ENDPOINT
      value: "<S3_ENDPOINT>"
    - name: S3_BUCKET
      value: "<S3_BUCKET>"
```

Configuração automática (initialData)

Habilite esta opção para provisionar o CloudBeaver automaticamente, já com administrador e configurações iniciais pré-definidas, sem necessidade de assistente de configuração. Esse modo é indicado para GitOps, ambientes descartáveis e instalações sem intervenção manual.

O workspace deve estar vazio na primeira inicialização. Não reutilize o mesmo PVC de uma instalação interativa anterior sem limpeza ou ajuste prévio, pois os dados persistidos podem conflitar com a configuração automática.

Como funciona

A configuração automática utiliza um arquivo `initial-data.conf` montado no container do CloudBeaver em `/opt/cloudbeaver/conf/`. Esse arquivo contém as configurações de administrador que o CloudBeaver lê na primeira inicialização, quando o workspace está vazio. O arquivo é montado no container por `extraVolumes` e `extraVolumeMounts`.

Para habilitar esse modo, defina `initialData.enabled: true` e configure o Secret do administrador antes da instalação:

© Copyright Tecnisys 2026. All rights reserved.

```yaml
tdp-cloudbeaver:
  initialData:
    enabled: true
    name: tdp-cloudbeaver-initial-data
    adminName: cbadmin  # Não pode ser "admin"

  adminSecret:
    enabled: true
    name: tdp-cloudbeaver-admin-secret
    data:
      password: "<PASSWORD>"  # Nunca commite a senha em repositórios Git

  # Monta o arquivo initial-data.conf no container
  extraVolumes:
    - name: initial-data
      configMap:
        name: tdp-cloudbeaver-initial-data

  extraVolumeMounts:
    - name: initial-data
      mountPath: /opt/cloudbeaver/conf/initial-data.conf
      subPath: initial-data.conf
      readOnly: true
```

Implantar com configuração automática

`<RELEASE_NAME>` é o **nome do release** Helm (o primeiro argumento de `helm upgrade --install`; costuma ser o identificador do projeto ou do ambiente).
O caminho `.../charts/tdp-cloudbeaver` no URL OCI é o **nome do chart** no registry da Tecnisys — não o substitua pelo nome do projeto.
Use o mesmo `<NAMESPACE>` da sua instalação.

Instalação nova (workspace limpo)

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  --version <CHART_VERSION> \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

© Copyright Tecnisys 2026. All rights reserved.

EXEMPLO

Exemplo

```
helm upgrade --install cloudbeaver \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  --version 3.0.1 \
  -n tdp-project --create-namespace \
  -f values-cloudbeaver.yaml
```

Acessar o CloudBeaver

Terminal input

```
kubectl -n <NAMESPACE> port-forward svc/<SERVICE_NAME> 8978:8978
```

A porta **8978** é a porta padrão da interface web do CloudBeaver.

Abra `http://localhost:8978` e autentique-se com:

- **Usuário:** `cbadmin` (valor padrão de `initialData.adminName`; altere no seu arquivo de valores se necessário)
- **Senha:** a senha configurada em `adminSecret.data.password`

AVISOS IMPORTANTES

Workspace limpo obrigatório: a configuração automática só vale na primeira inicialização com workspace vazio.

Se o CloudBeaver já tiver sido configurado antes, remova o release e identifique o PVC de workspace antes de reinstalar:

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
kubectl get pvc -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Remova apenas o PVC usado pelo workspace do CloudBeaver:

Terminal input

```
kubectl delete pvc -n <NAMESPACE> <PVC_NAME>
```

Não use `kubectl delete pvc --all` em namespaces compartilhados, pois esse comando remove todos os PVCs do namespace.

Para o comportamento exato de PVCs e rótulos, consulte a documentação do chart ou `helm show values`.

Validação após initialData

1. **Confirmar se o ConfigMap está montado:**

Terminal input

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- ls -la /opt/cloudbeaver/conf/
kubectl exec -n <NAMESPACE> <POD_NAME> -- cat /opt/cloudbeaver/conf/initial-
data.conf
```

2. **Confirmar se o *workspace* está limpo:**

Terminal input

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- ls -la
/opt/cloudbeaver/workspace/.data/
```

Se existir `.cloudbeaver.runtime.conf`, o *workspace* não está limpo.
Remova o release, identifique o PVC do *workspace* e reinstale:

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl get pvc -n <NAMESPACE>
```

Remova apenas o PVC do *workspace* do CloudBeaver e volte a instalar:

Terminal input

```
kubectl delete pvc -n <NAMESPACE> <PVC_NAME>
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f meu-values.yaml
```

Não use `kubectl delete pvc --all` em namespaces compartilhados.

3. **Conteúdo do ConfigMap:**

Terminal input

```
kubectl get configmap <CONFIGMAP_NAME> -n <NAMESPACE> -o yaml
```

Confirme que o arquivo contém o formato JSON correto com `adminName` e `adminPassword`.

4. **Logs do CloudBeaver:**

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep
-i config
```

Procure mensagens sobre modo de configuração ou criação do administrador.

Combinando arquivos de valores

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> \
  -f meu-values.yaml \
  -f values-integracao.yaml
```

Esta abordagem permite separar configuração base e integrações em arquivos distintos para melhor legibilidade e reutilização.

Padrões Avançados

Múltiplas instâncias do CloudBeaver

Você pode instalar múltiplas instâncias em namespaces distintos (cada combinação `<RELEASE_NAME>` + `<NAMESPACE>` + `<VALUES_FILE>` representa um ambiente):

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

EXEMPLO

Dois ambientes ilustrativos com nomes fixos:

Exemplo

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install cloudbeaver-dev \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n cloudbeaver-dev --create-namespace \
  -f values-dev.yaml

helm upgrade --install cloudbeaver-prod \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n cloudbeaver-prod --create-namespace \
  -f values-production.yaml
```

Compartilhar Secrets entre ClickHouse e CloudBeaver

Para sincronizar a mesma senha entre ClickHouse e CloudBeaver, use um Secret compartilhado (`<SECRET_NAME>`) e releases distintos (`<CLICKHOUSE_RELEASE>`, `<CLOUDBEAVER_RELEASE>`).

1. Criar Secret compartilhado:

Terminal input

```
# Defina a senha no shell (não commite em repositório)
export CLICKHOUSE_PASSWORD='<PASSWORD>'

kubectl create secret generic <SECRET_NAME> \
  --from-literal=password="${CLICKHOUSE_PASSWORD}" \
  -n <NAMESPACE>
```

EXEMPLO

Exemplo

```
export CLICKHOUSE_PASSWORD='MeuSenhaSegura123!'

kubectl create secret generic tdp-shared-clickhouse-password \
  --from-literal=password="${CLICKHOUSE_PASSWORD}" \
  -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

2. Configurar ClickHouse para usar a senha:

Terminal input

```
helm upgrade --install <CLICKHOUSE_RELEASE> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> --create-namespace \
  --set tdp-clickhouse.clickhouse.defaultUser.password="${CLICKHOUSE_PASSWORD}"
```

EXEMPLO

Exemplo

```
helm upgrade --install tdp-clickhouse \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> --create-namespace \
  --set tdp-
clickhouse.clickhouse.defaultUser.password="${CLICKHOUSE_PASSWORD}"
```

3. Configurar CloudBeaver para referenciar o mesmo Secret:

```
tdp-cloudbeaver:
  dataSources:
    enabled: true
    enabledClickhouse: true
    clickhouseHost: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
    clickhousePort: 8123
    database: "default"
    user: "default"

    # Referenciar o Secret compartilhado
    passwordSecret:
      name: <SECRET_NAME>
      key: password

    secret:
      enabled: false
```

© Copyright Tecnisys 2026. All rights reserved.

EXEMPLO

Exemplo

```
tdp-cloudbeaver:
  dataSources:
    passwordSecret:
      name: tdp-shared-clickhouse-password
      key: password
    secret:
      enabled: false
```

4. Implantar CloudBeaver:

Terminal input

```
helm upgrade --install <CLOUDBEAVER_RELEASE> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

EXEMPLO

Exemplo

```
helm upgrade --install tdp-cloudbeaver \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver \
  -n <NAMESPACE> --create-namespace \
  -f values.yaml
```

Rotação de senhas coordenada

Para rotacionar a senha mantendo ambos os componentes sincronizados:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
# 1. Atualizar Secret compartilhado
export NEW_PASSWORD='<PASSWORD>'

kubectl create secret generic <SECRET_NAME> \
  --from-literal=password="${NEW_PASSWORD}" \
  -n <NAMESPACE> \
  --dry-run=client -o yaml | kubectl apply -f -

# 2. Atualizar ClickHouse
helm upgrade <CLICKHOUSE_RELEASE> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> \
  --set tdp-clickhouse.clickhouse.defaultUser.password="${NEW_PASSWORD}"

# 3. Aguardar ClickHouse estar pronto
kubectl rollout status statefulset/<STATEFULSET_NAME> -n <NAMESPACE>

# 4. Remover pod do CloudBeaver para forçar atualização
kubectl delete pod -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver

# 5. Verificar conexão
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver | grep -i
clickhouse
```

EXEMPLO

Exemplo

```
export NEW_PASSWORD='NovaSenhaSegura456!'

kubectl create secret generic tdp-shared-clickhouse-password \
  --from-literal=password="${NEW_PASSWORD}" \
  -n <NAMESPACE> \
  --dry-run=client -o yaml | kubectl apply -f -

helm upgrade tdp-clickhouse \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse \
  -n <NAMESPACE> \
  --set tdp-
clickhouse.clickhouse.defaultUser.password="${NEW_PASSWORD}"

kubectl rollout status statefulset/tdp-clickhouse -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl delete pod -n <NAMESPACE> -l app.kubernetes.io/name=tdp-
cloudbeaver

kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=tdp-cloudbeaver |
grep -i clickhouse
```

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake

CHART

Version 3.0.1 | Type application | AppVersion 4.0.1

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado

O que é o Delta Lake?

O Delta Lake é um formato de tabela aberta que adiciona transações ACID ao armazenamento de objetos (S3/Ozone). Em vez de gravar arquivos Parquet diretamente, o Delta Lake mantém um **transaction log** — um histórico de todas as operações (INSERT, UPDATE, DELETE, MERGE) realizadas na tabela.

Com o tempo, esse modelo gera dois tipos de acúmulo que precisam de limpeza periódica:

1. **Arquivos órfãos**: arquivos Parquet que existem no S3 mas não são mais referenciados por nenhuma versão ativa da tabela
2. **Versões antigas**: o transaction log mantém histórico indefinidamente; sem limpeza, ele cresce sem limite

O chart `tdp-deltalake` não é uma aplicação contínua — ele cria **Kubernetes CronJobs** que rodam periodicamente para realizar essa manutenção automaticamente, utilizando Apache Spark 4.0.2.

> **PARA SABER MAIS**
>
> Consulte Delta Lake — Conceitos para uma visão completa do formato, transações ACID e casos de uso.

© Copyright Tecnisys 2026. All rights reserved.

Jobs de manutenção fornecidos

Job	Descrição
VACUUM	Remove arquivos antigos não referenciados pelo transaction log
OPTIMIZE	Compacta arquivos pequenos em arquivos maiores
OPTIMIZE Z-ORDER	Otimiza o layout dos dados para consultas com filtros específicos
DESCRIBE HISTORY	Registra histórico de alterações para auditoria

Estrutura de valores (Helm)

O chart `tdp-deltalake` distribui a configuração em dois blocos:

- **`maintenance:`** — jobs de manutenção (VACUUM, OPTIMIZE, Z-ORDER, DESCRIBE HISTORY), agendamento (cron), retenção e configuração Spark dos jobs.
- **`spark:`** — valores do chart upstream do Spark: imagem base, labels, configuração de master e workers.

```
maintenance:
enabled: true
spark:
  enabled: true
  image:
    repository: "registry.tecnisys.com.br/tdp-dev/images/spark"
    tag: "4.0.2-0"
jobs:
  vacuum:
    enabled: true
    schedule: "0 2 * * *"
    retentionHours: 168

spark:
image:
```

© Copyright Tecnisys 2026. All rights reserved.

```
registry: registry.tecnisys.com.br
repository: tdp-dev/images/spark
tag: 4.0.2-0
```

Visão geral

Propriedade	Valor
Chart	`tdp-deltalake`
Tipo	`maintenance` (CronJobs)
Versão do Delta Lake	4.0.1
Versão do chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake`
Spark	4.0.2
Scala	2.13

Páginas relacionadas

- Integrações — Delta Lake: S3/MinIO, Trino, Spark, Airflow.
- Segurança — Delta Lake: Secret de credenciais S3 e boas práticas.

Pré-requisitos

- **Kubernetes 1.32+**, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- **Helm 3.2.0+**
- Secret `s3-credentials` com as chaves `access-key` e `secret-key` criado no namespace antes do deploy. Consulte Segurança — Delta Lake para instruções.

Instalação

Instalação

© Copyright Tecnisys 2026. All rights reserved.



Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake \
  --version <CHART_VERSION> \
  -n <NAMESPACE> --create-namespace
```

Placeholder	Descrição
`<RELEASE_NAME>`	Nome do release Helm
`<NAMESPACE>`	Namespace Kubernetes de instalação
`<CHART_VERSION>`	Versão do chart

OpenShift

No OpenShift, o chart aplica `force` no Security Context por padrão:

```
global:
  compatibility:
    openshift:
      adaptSecurityContext: force
```

Esse valor já vem pré-configurado no `values.yaml`. A justificativa desse padrão está na página Configuração Geral, aba **Decisões de ambiente que afetam a configuração**, no tópico **Compatibilidade do orquestrador e ajustes de segurança**.

Verificação pós-instalação

Terminal input

```
# Listar CronJobs criados
kubectl get cronjobs -n <NAMESPACE>

# Verificar últimas execuções
kubectl get jobs -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
# Logs de um job específico
kubectl logs -n <NAMESPACE> -l job-name=<JOB_NAME>
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão
`maintenance.enabled`	Habilitar jobs de manutenção	`true`
`maintenance.spark.enabled`	Habilitar dependência Spark	`true`
`maintenance.spark.image.repository`	Imagem Spark	registry.tecnisys.com.br/tdp-dev/images/spark
`maintenance.spark.image.tag`	Tag da imagem Spark	4.0.2-0
`maintenance.spark.resources.requests.cpu`	CPU mínima por job	`1`
`maintenance.spark.resources.requests.memory`	Memória mínima por job	`2Gi`
`maintenance.spark.resources.limits.cpu`	CPU máxima por job	`2`
`maintenance.spark.resources.limits.memory`	Memória máxima por job	`4Gi`
`maintenance.jobs.vacuum.enabled`	Habilitar job VACUUM	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`maintenance.jobs.vacuum.schedule`	Cron do VACUUM	`0 2 * * *`
`maintenance.jobs.vacuum.retentionHours`	Retenção VACUUM (horas)	`168`
`maintenance.jobs.optimize.enabled`	Habilitar job OPTIMIZE	`true`
`maintenance.jobs.optimize.schedule`	Cron do OPTIMIZE	`0 3 * * 0`
`maintenance.jobs.optimizeZOrder.enabled`	Habilitar OPTIMIZE Z-ORDER	`false`
`maintenance.jobs.optimizeZOrder.schedule`	Cron do OPTIMIZE Z-ORDER	`0 1 * * 6`
`maintenance.jobs.describeHistory.enabled`	Habilitar DESCRIBE HISTORY	`false`
`maintenance.jobs.describeHistory.schedule`	Cron do DESCRIBE HISTORY	`0 6 * * *`
`maintenance.defaultTablePath`	Caminho padrão das tabelas Delta no S3	`s3a://warehouse/delta`

Detalhes de configuração

Jobs de manutenção

Configure os jobs em `maintenance.jobs`:

© Copyright Tecnisys 2026. All rights reserved.

```
maintenance:
  enabled: true
  jobs:
    vacuum:
      enabled: true
      schedule: "0 2 * * *"    # Diariamente às 2h
      retentionHours: 168      # 7 dias (padrão)

    optimize:
      enabled: true
      schedule: "0 3 * * 0"    # Semanalmente aos domingos às 3h

    optimizeZOrder:
      enabled: false
      schedule: "0 1 * * 6"    # Sábados às 1h

    describeHistory:
      enabled: false
      schedule: "0 6 * * *"    # Diariamente às 6h
```

Cada `schedule` define **quando** o cluster executa trabalho intensivo em I/O contra o *warehouse*. Em janelas de pico de consultas, VACUUM e OPTIMIZE competem por largura de banda com Trino, Spark ou outros consumidores. Ajuste os horários para janelas de baixo uso.

O `retentionHours` do VACUUM está ligado ao tempo mínimo em que versões antigas precisam estar visíveis para *time travel* e leituras longas. Alterar a retenção sem alinhar com o uso real pode causar erros de leitura ou custo excessivo de storage.

VACUUM

Remove arquivos que não são mais referenciados pelo Delta Log:

```
VACUUM delta.`s3a://warehouse/delta/tabela` RETAIN 168 HOURS;
```

> **ATENÇÃO**
>
> O período de retenção padrão é 7 dias (168 horas). Não reduza abaixo de 7 dias em produção — isso pode causar perda de dados para consultas em

© Copyright Tecnisys 2026. All rights reserved.

andamento.

OPTIMIZE

Compacta arquivos pequenos em arquivos maiores para melhorar a performance de leitura:

```
OPTIMIZE delta.`s3a://warehouse/delta/tabela`;
```

OPTIMIZE com Z-ORDER

Otimiza o layout dos dados para consultas com filtros específicos:

```
OPTIMIZE delta.`s3a://warehouse/delta/tabela` ZORDER BY (coluna1, coluna2);
```

Configuração do Spark

A imagem e os recursos definem o teto de CPU e memória que cada CronJob pode usar ao executar operações sobre tabelas grandes. Sintomas como *OOMKilled* ou execuções muito longas costumam levar ao aumento de `limits`. A tag deve corresponder ao Spark 4.0.2 e Delta 4.0.1 indicados na compatibilidade.

```
maintenance:
spark:
  image:
    repository: "registry.tecnisys.com.br/tdp-dev/images/spark"
    tag: "4.0.2-0"
  resources:
    requests:
      cpu: 1
      memory: 2Gi
    limits:
      cpu: 2
      memory: 4Gi
  config:
    "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>.
<NAMESPACE>.svc.cluster.local:9000"
    "spark.hadoop.fs.s3a.path.style.access": "true"
```

© Copyright Tecnisys 2026. All rights reserved.

Integrações

Esta página cobre os CronJobs de manutenção. A página Integrações — Delta Lake concentra as credenciais de armazenamento (S3/MinIO) e como Trino, Spark ou Airflow consomem as mesmas tabelas Delta.

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Este comando remove todos os recursos do release (CronJobs, Jobs, ServiceAccount, RBAC). Este chart não cria PVCs.

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 4.0.1

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado

O chart `tdp-deltalake` executa jobs de manutenção (VACUUM, OPTIMIZE) no armazenamento S3/MinIO. As credenciais de acesso ao bucket são fornecidas via Secret Kubernetes `s3-credentials`, lido pelos CronJobs via `env.secretKeyRef`.

Credenciais e Secrets

Crie o Secret no namespace antes do deploy:

```
kubectl -n <NAMESPACE> create secret generic s3-credentials \
  --from-literal=access-key="<ACCESS_KEY>" \
  --from-literal=secret-key="<SECRET_KEY>"
```

Chave	Descrição
`access-key`	Chave de acesso ao bucket S3/MinIO
`secret-key`	Chave secreta para autenticação S3

O nome do Secret deve ser exatamente `s3-credentials`. Os CronJobs do chart referenciam esse nome diretamente.

Configuração do endpoint S3

© Copyright Tecnisys 2026. All rights reserved.



O endpoint do S3/MinIO é configurado no bloco `maintenance.spark.config`:

```
maintenance:
  spark:
    config:
      "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>:<S3_PORT>"
      "spark.hadoop.fs.s3a.path.style.access": "true"
```

Substitua `<S3_ENDPOINT>` pelo endereço interno do serviço S3 no cluster. Para o Ozone instalado junto ao TDP, o endereço padrão segue o padrão:

```
http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878
```

Boas práticas

Aspecto	Recomendação
Credenciais	Não versionar `access-key` e `secret-key` em repositório Git
Rotação	Recriar o Secret e aguardar o próximo ciclo de execução dos CronJobs
Acesso mínimo	Configurar credenciais com permissão apenas nos buckets usados pelos jobs de manutenção

Solução de problemas

Problema	Causa provável	Solução
CronJob falha com erro de autenticação	Secret ausente ou chaves com nome incorreto	Verificar Secret `s3-credentials` no namespace
Endpoint S3 inacessível	URL ou porta incorretos	Testar conectividade com `kubectl exec` dentro do

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
		namespace
Permissão negada no bucket	Credenciais sem acesso ao path configurado	Revisar permissões no MinIO/Ozone

Para a lista completa de parâmetros, use `helm show values` na versão do chart que instalou.

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake – Integração

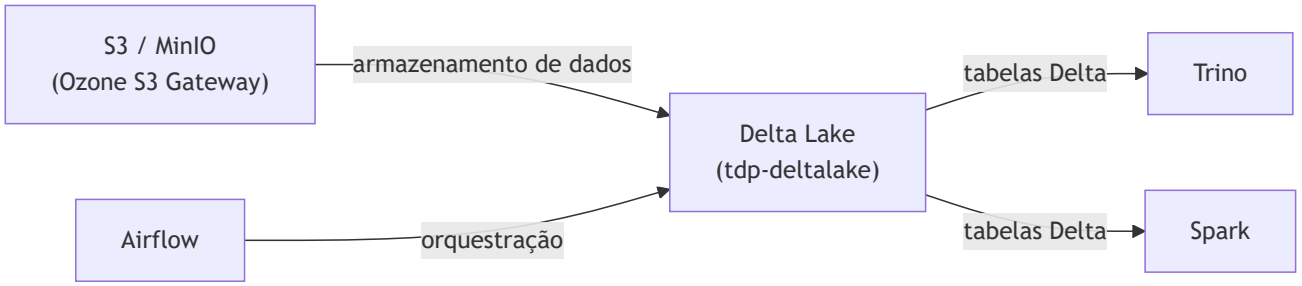
CHART

Version 3.0.1 | Type application | AppVersion 4.0.1

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O chart `tdp-deltalake` requer acesso ao armazenamento S3-compatível (Apache Ozone S3 Gateway, MinIO ou outro endpoint S3) e pode ser integrado às principais ferramentas de processamento e orquestração do TDP.

S3 / MinIO

Os CronJobs de manutenção requerem o Secret `s3-credentials` com as credenciais de acesso ao S3. Crie-o antes do deploy conforme descrito em Segurança — Delta Lake.

Configurar o endpoint S3

```
maintenance:
  spark:
    config:
      "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>.
<NAMESPACE>.svc.cluster.local:9000"
      "spark.hadoop.fs.s3a.path.style.access": "true"
```

Parâmetros S3

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Exemplo
`spark.hadoop.fs.s3a.endpoint`	URL do endpoint S3	`http://ozone-s3g.<NAMESPACE>.svc.cluster.local:9000`
`spark.hadoop.fs.s3a.path.style.access`	Forçar path-style (necessário para MinIO/Ozone)	`"true"`

Trino

O Trino pode consultar tabelas Delta Lake via conector Delta. A configuração é realizada do lado do chart `tdp-trino` — consulte Configuração do Trino.

Spark

As tabelas Delta Lake podem ser processadas diretamente pelo Spark. A configuração é realizada do lado do chart `tdp-spark` — consulte Configuração do Spark.

Airflow

O Airflow pode orquestrar pipelines que leem e escrevem tabelas Delta Lake. A configuração de conexão é realizada do lado do chart `tdp-airflow` — consulte Integrações — Airflow.

Combinando arquivos de valores

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake \
  -n <NAMESPACE> \
  -f meu-values.yaml \
  -f values-integracao.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

Hive Metastore

CHART

Version 3.0.1 | Type application | AppVersion 4.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado | External DB suportado

O que é o Hive Metastore?

O chart `tdp-hive-metastore` instala o **Apache Hive Metastore** 4.0.0 no Kubernetes e entrega um serviço interno de catálogo, acessado por endpoint Thrift na porta `9083`.

Esse catálogo guarda metadados de bancos, tabelas, colunas, tipos, partições e localizações. Os dados das tabelas continuam no warehouse configurado; o Metastore registra como esses dados devem ser encontrados e interpretados pelos clientes.

Na prática, quando um cliente compatível quer ler uma tabela chamada `vendas`, por exemplo, ele consulta o Metastore para obter o esquema da tabela, as partições disponíveis e o caminho dos arquivos. Hive, Spark, Trino e Iceberg são exemplos de clientes que podem consumir esse catálogo.

No TDP, o chart suporta warehouse em modo `file` para cenários simples ou validações, e em modo `s3` para integração com armazenamento de objetos S3-compatível, como o Ozone. Os metadados do catálogo ficam em PostgreSQL, interno ou externo.

> **PARA SABER MAIS**
>
> Consulte Apache Hive — Conceitos, seção Hive Metastore para entender o papel do catálogo de metadados no ecossistema de dados.

© Copyright Tecnisys 2026. All rights reserved.

Como o Hive Metastore se encaixa no TDP

```
Clientes compatíveis
(Hive, Spark, Trino, Iceberg...)
     |
     |  "Onde está a tabela 'vendas'?"
     v
Hive Metastore  -->  PostgreSQL (armazena metadados)
     |
     |  "Está no warehouse configurado"
     v
Warehouse file ou S3/Ozone (onde os dados de fato estão)
```

O Hive Metastore opera como serviço de infraestrutura do plano de dados. Em vez de expor uma interface web para usuários finais, ele atende clientes internos que precisam consultar ou atualizar metadados de tabelas.

Por isso, a configuração normalmente se concentra no Service interno, no PostgreSQL de metadados e no modo de warehouse. O consumo acontece pelo endpoint Thrift do Service na porta `9083`.

Componentes implantados

Componente	Descrição
Metastore Server	Serviço Thrift na porta 9083 que responde a consultas de metadados
PostgreSQL (interno ou externo)	Banco que armazena esquemas, tabelas, partições e localizações
Jobs auxiliares TDP	Jobs de validação/criação de banco e limpeza de recursos auxiliares

Estrutura de valores (Helm)

O chart `tdp-hive-metastore` distribui a configuração em dois blocos:

© Copyright Tecnisys 2026. All rights reserved.

- `tdp-hive-metastore:` — configuração principal: imagem, PostgreSQL (embutido por padrão ou externo), tipo de warehouse (`file` ou `s3`), recursos, segurança.
- `TDP-Settings:` — configuração opcional usada apenas quando `postgres.enabled: false`; habilita os hooks TDP de validação, provisionamento e autenticação no PostgreSQL externo.

```
TDP-Settings:
  externalDatabase:
    enabled: false

tdp-hive-metastore:
  postgres:
    enabled: true
  metastore:
    type: file
```

Visão geral

Propriedade	Valor
Chart	`tdp-hive-metastore`
Versão do Hive Metastore	4.0.0
Versão do chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore`
Porta Thrift	`9083`
Exposição HTTP externa	Não se aplica (acesso via ClusterIP, sem Ingress)

Páginas relacionadas

- Integração — Hive Metastore: URI Thrift e exemplos de consumo por clientes compatíveis.

© Copyright Tecnisys 2026. All rights reserved.

- Segurança — Hive Metastore: Secrets, credenciais do banco e cuidados com senhas.

Pré-requisitos

- **Kubernetes 1.32+**, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- **Helm 3.2.0+**
- Registry OCI da Tecnisys acessível pelo ambiente de instalação
- Se usar PostgreSQL externo: instância acessível a partir do cluster, `max_connections` ≥ 300, e Secret administrativo no padrão esperado pelo chart

Instalação

Instalação

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore \
  --version <CHART_VERSION> \
  -n <NAMESPACE> --create-namespace
```

Placeholder	Descrição
`<RELEASE_NAME>`	Nome do release Helm
`<NAMESPACE>`	Namespace Kubernetes de instalação
`<CHART_VERSION>`	Versão do chart

Verificar instalação

Terminal input

```
kubectl -n <NAMESPACE> get all
```

© Copyright Tecnisys 2026. All rights reserved.

Espera-se Deployment/Service do metastore (e do PostgreSQL interno, se `postgres.enabled: true`), com nomes derivados de `<RELEASE_NAME>`.

Validação do Service do Metastore

O chart publica o Metastore por um Service interno na porta `9083`. Após a instalação, valide se o Service possui endpoints antes de configurar clientes consumidores.

Terminal input

```
kubectl get endpoints -n <NAMESPACE> <RELEASE_NAME>-metastore
```

Se aparecer `<none>`, aguarde a conclusão do deploy e verifique os pods do Metastore. A página Integração — Hive Metastore mostra como usar esse endpoint nos consumidores.

Teste local:

Terminal input

```
kubectl port-forward -n <NAMESPACE> svc/<RELEASE_NAME>-metastore 9083:9083
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Valor por omissão / exemplo
`tdp-hive-metastore.postgres.enabled`	PostgreSQL interno	`true`
`tdp-hive-metastore.postgres.internal.pvcSize`	Tamanho do PVC do PostgreSQL interno (não definido em `values.yaml` — exemplo recomendado)	`1Gi`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor por omissão / exemplo
`tdp-hive-metastore.postgres.internal.maxConnections`	max_connections do PostgreSQL interno (não definido em `values.yaml` — exemplo recomendado)	`500`
`tdp-hive-metastore.postgres.external.host`	Host PostgreSQL externo	`""`
`tdp-hive-metastore.postgres.external.port`	Porta PostgreSQL externo	`5432`
`tdp-hive-metastore.postgres.database`	Nome do banco de dados	`hive`
`tdp-hive-metastore.postgres.username`	Usuário do banco	`hive`
`tdp-hive-metastore.postgres.password`	Senha do banco (usar Secret em produção)	`<POSTGRES_PASSWORD>`
`tdp-hive-metastore.metastore.type`	Tipo de metastore (`file` ou `s3`)	`file`
`tdp-hive-metastore.metastore.image`	Imagem do Metastore	`registry.tecnisys.com.br/tdp-dev/images/hive-metastore:4.0.0-0`
`tdp-hive-metastore.metastore.file.warehouseDir`	Diretório de warehouse no modo `file`	`/opt/hive/warehouse`
`tdp-hive-metastore.metastore.s3.warehouseDir`	Caminho S3 do warehouse	`s3a://warehouse/hive`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor por omissão / exemplo
`tdp-hive-metastore.metastore.s3.endpoint`	Endpoint S3	`""`
`tdp-hive-metastore.metastore.s3.bucket`	Bucket do warehouse	`warehouse`
`TDP-Settings.externalDatabase.enabled`	Habilitar hooks de banco externo	`false`
`TDP-Settings.externalDatabase.recreate`	Recriar banco/usuário no deploy	`false`
`TDP-Settings.externalDatabase.externalSecret.releaseName`	Release do PostgreSQL (Secret com `postgres-password`)	`tdp-postgresql-project`

Detalhes de configuração

Banco de dados (PostgreSQL)

O Hive Metastore armazena os metadados do catálogo em PostgreSQL. A escolha entre PostgreSQL embutido e externo é uma das primeiras decisões de configuração.

Para entender quando usar cada um, consulte PostgreSQL interno versus externo na página de Configuração Geral.

Requisito	Mínimo	Recomendado	Exemplo (TDP PostgreSQL)
max_connections	300	400+	400
Operações concorrentes	50+	100+	Suportado pelo padrão TDP

© Copyright Tecnisys 2026. All rights reserved.

PostgreSQL embutido (padrão)

Com `postgres.enabled: true`, o chart provisiona um PostgreSQL dedicado ao metastore. O PVC associado armazena **somente** metadados relacionais (nomes de tabelas, partições, URIs no objeto), enquanto os dados analíticos permanecem no Ozone/S3. Dimensione disco e backup desse PVC conforme o crescimento do **catálogo** (quantidade de tabelas e partições), não o volume dos arquivos de dados no bucket.

O bloco `postgres.internal.*` não vem definido em `values.yaml` — defina explicitamente os valores desejados, conforme o exemplo abaixo:

```
tdp-hive-metastore:
  postgres:
    enabled: true

    internal:
      image: "registry.tecnisys.com.br/community/images/postgres:15"
      pvcSize: 1Gi
      maxConnections: 500
      sharedBuffers: 512MB
      effectiveCacheSize: 1.2GB
      maintenanceWorkMem: 128MB
      workMem: 32MB
      walBuffers: 16MB

    database: hive
    username: hive
    password: "<PASSWORD>"
```

PostgreSQL externo

Com PostgreSQL externo, a disponibilidade e o dimensionamento do metastore passam a depender da instância compartilhada. O armazenamento de objetos onde ficam os arquivos das tabelas **não** migra com a troca do host do Postgres — alterar só o banco muda onde ficam os registros que apontam para o *warehouse*, não o conteúdo do *bucket*.

Defina `tdp-hive-metastore.postgres.enabled: false`, configure `tdp-hive-metastore.postgres.external.*` e habilite `TDP-Settings.externalDatabase.enabled: true` para que os hooks TDP validem/provisionem o banco externo.

© Copyright Tecnisys 2026. All rights reserved.

```
TDP-Settings:
  externalDatabase:
    enabled: true
    recreate: false
    externalSecret:
      releaseName: "<POSTGRESQL_RELEASE>"

tdp-hive-metastore:
  postgres:
    enabled: false
    external:
      host: "<POSTGRESQL_SERVICE>"
      port: 5432
```

Os jobs de *hook* reutilizam `TDP-Settings.externalDatabase`: autenticam no PostgreSQL externo usando o Secret associado ao nome em `externalSecret.releaseName` (chave `postgres-password`, no padrão TDP).

- Use `recreate: true` apenas quando quiser recriar banco/usuário Hive de forma explícita; para *upgrades* que preservem dados, mantenha `recreate: false` (valor padrão).
- Para detalhes de Secret e credenciais, consulte Segurança — Hive Metastore.

Sobrescritas via CLI:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore \
  -n <NAMESPACE> --create-namespace \
  --set TDP-Settings.externalDatabase.recreate=false \
  --set TDP-Settings.externalDatabase.externalSecret.releaseName=
<POSTGRESQL_RELEASE>
```

Modo de armazenamento do warehouse

O parâmetro `tdp-hive-metastore.metastore.type` define como o Metastore referencia a área de warehouse:

© Copyright Tecnisys 2026. All rights reserved.

Modo	Quando usar	Configuração principal
`file`	Ambientes simples ou validação inicial	`tdp-hive-metastore.metastore.file.warehouseDir`
`s3`	Integração com armazenamento S3-compatível, como Ozone	`tdp-hive-metastore.metastore.s3.*`

Exemplo resumido para warehouse S3:

```
tdp-hive-metastore:
  metastore:
    type: s3
    s3:
      warehouseDir: s3a://warehouse/hive
      bucket: warehouse
      endpoint: "http://<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local:9000"
      accessKey: "<S3_ACCESS_KEY>"
      secretKey: "<S3_SECRET_KEY>"
      pathStyleAccess: true
      region: us-east-1
```

Os detalhes de consumo por Spark, Trino, Iceberg e outros componentes ficam na página Integração — Hive Metastore.

Recursos do Metastore

Valores típicos de referência: memória 1Gi *request* / 2Gi *limit*, CPU 500m *request* / 1000m *limit*, volume persistente configurável. Confirme com `helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore`.

Solução de problemas

Solução de problemas

1. **Esgotamento de pool de conexões**: `max_connections` ≥ 300 no PostgreSQL.
2. **Autenticação**: Secrets e montagens corretas.

© Copyright Tecnisys 2026. All rights reserved.

3. **Storage**: permissões do PV.
4. **Rede**: serviço PostgreSQL alcançável.
5. **Endpoint Thrift**: Service do Metastore com endpoints na porta `9083`.

Comandos úteis (adaptando nomes de release/pod):

Terminal input

```
kubectl -n <NAMESPACE> exec <POSTGRESQL_POD> -- psql -c "SHOW max_connections;"
kubectl -n <NAMESPACE> exec <POSTGRESQL_POD> -- psql -c "SELECT count(*) FROM
pg_stat_activity;"
kubectl -n <NAMESPACE> logs deployment/<RELEASE_NAME>-metastore
```

Para validar credenciais diretamente no banco Hive, use os comandos da página Segurança — Hive Metastore.

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Este comando remove os recursos do release (Deployments, Services, ConfigMaps, Jobs). Os PVCs do PostgreSQL interno são **mantidos por padrão** para preservar os metadados do catálogo.

Para remover os PVCs do PostgreSQL interno (operação destrutiva e irreversível):

Terminal input

```
kubectl delete pvc -n <NAMESPACE> -l app.kubernetes.io/name=postgresql
```

WARNING

© Copyright Tecnisys 2026. All rights reserved.



A remoção dos PVCs elimina permanentemente todos os metadados do catálogo Hive (mapeamento de tabelas, esquemas, partições). Os dados nos buckets S3/Ozone **não** são afetados, mas os clientes perderão acesso às tabelas pelos seus nomes até que o catálogo seja recriado.

© Copyright Tecnisys 2026. All rights reserved.

Hive Metastore – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 4.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado | External DB suportado

No Hive Metastore, os cuidados de segurança se concentram nas credenciais usadas para PostgreSQL e S3, nos Secrets associados ao deploy e nos contextos de segurança aplicados aos pods e containers.

Configurações de autenticação de usuários, SSO, LDAP, Ranger e políticas de autorização ficam fora do escopo desta página.

Credenciais e Secrets

O Hive Metastore usa credenciais para acessar o PostgreSQL que armazena os metadados e, quando `metastore.type: s3`, para acessar o endpoint S3/S3A configurado no chart.

Não armazene senhas em arquivos `values.yaml` versionados. Use arquivos de valores privados, CI seguro ou mecanismo de gestão de Secrets do ambiente.

Credenciais PostgreSQL

Com `postgres.enabled: true`, o chart usa PostgreSQL interno. Defina a senha do usuário `hive` em `tdp-hive-metastore.postgres.password`:

```
tdp-hive-metastore:
  postgres:
    enabled: true
    database: hive
```

© Copyright Tecnisys 2026. All rights reserved.

```
    username: hive
    password: "<POSTGRES_PASSWORD>"
```

Se precisar conectar diretamente ao banco `hive`, recupere a senha gerada no Secret do release:

```
kubectl get secret -n <NAMESPACE> <RELEASE_NAME>-hive-database \
  -o jsonpath="{.data.password}" | base64 --decode
```

Com PostgreSQL externo (`postgres.enabled: false`), os hooks de pré-instalação buscam a senha de administração no Secret associado ao release definido em `TDP-Settings.externalDatabase.externalSecret.releaseName`. Esse Secret deve conter a chave `postgres-password` no padrão TDP. Para os parâmetros de configuração do banco externo, consulte Configuração do Hive Metastore.

Credenciais S3

Para usar o warehouse em modo S3, configure `metastore.type: s3` e informe as credenciais em `tdp-hive-metastore.metastore.s3.accessKey` e `tdp-hive-metastore.metastore.s3.secretKey`:

```
tdp-hive-metastore:
  metastore:
    type: s3
    s3:
      endpoint: "http://<S3_SERVICE>.<NAMESPACE>.svc.cluster.local:9000"
      accessKey: "<S3_ACCESS_KEY>"
      secretKey: "<S3_SECRET_KEY>"
```

Proteja esses valores por meio de arquivo de valores privado, CI seguro ou ferramenta de gestão de Secrets adotada no ambiente. Para configurar o modo `s3`, consulte a página de Configuração do Hive Metastore.

imagePullSecrets

Use `tdp-hive-metastore.imagePullSecrets` para baixar imagens a partir de registries privados:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-hive-metastore:
  imagePullSecrets:
    - name: tdp-registry
```

O Secret referenciado deve existir no namespace em que o Hive Metastore será instalado. Em ambientes que usam política de cluster para injetar `imagePullSecrets`, alinhe este valor à estratégia adotada no cluster.

podSecurityContext e containerSecurityContext

Configure os contextos de segurança dos workloads e jobs auxiliares em `podSecurityContext` e `containerSecurityContext`:

```
tdp-hive-metastore:
  podSecurityContext:
    seccompProfile:
      type: RuntimeDefault

  containerSecurityContext:
    allowPrivilegeEscalation: false
    readOnlyRootFilesystem: false
    seccompProfile:
      type: RuntimeDefault
    capabilities:
      drop:
        - ALL
```

Esses valores reduzem permissões do container e aplicam `seccompProfile: RuntimeDefault`. Preserve esses padrões, salvo quando houver exigência explícita do ambiente Kubernetes/OpenShift.

Solução de problemas

Problema	Causa provável	Solução
Metastore não sobe (banco externo)	Secret com `postgres-password` ausente ou nome do release incorreto	Verificar `externalSecret.releaseName`

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
Falha nos hooks de pré-instalação	PostgreSQL não acessível ou `max_connections` insuficiente	Verificar conectividade e `SHOW max_connections`
Autenticação negada	Credenciais incorretas no values	Rever `postgres.username` e `postgres.password`
Credenciais S3 não funcionam	`accessKey`, `secretKey` ou endpoint S3 incorretos	Conferir `tdp-hive-metastore.metastore.s3.*` e evitar credenciais em valores versionados
Pull de imagem falha	Secret de registry ausente ou nome incorreto	Verificar `tdp-hive-metastore.imagePullSecrets` e se o Secret existe no namespace
Pod bloqueado por política de segurança	`podSecurityContext` ou `containerSecurityContext` incompatível com a política do cluster	Conferir eventos do Pod e ajustar apenas conforme exigência do ambiente

Para a lista completa de parâmetros, use `helm show values` na versão do chart que instalou.

© Copyright Tecnisys 2026. All rights reserved.

Hive Metastore – Integração

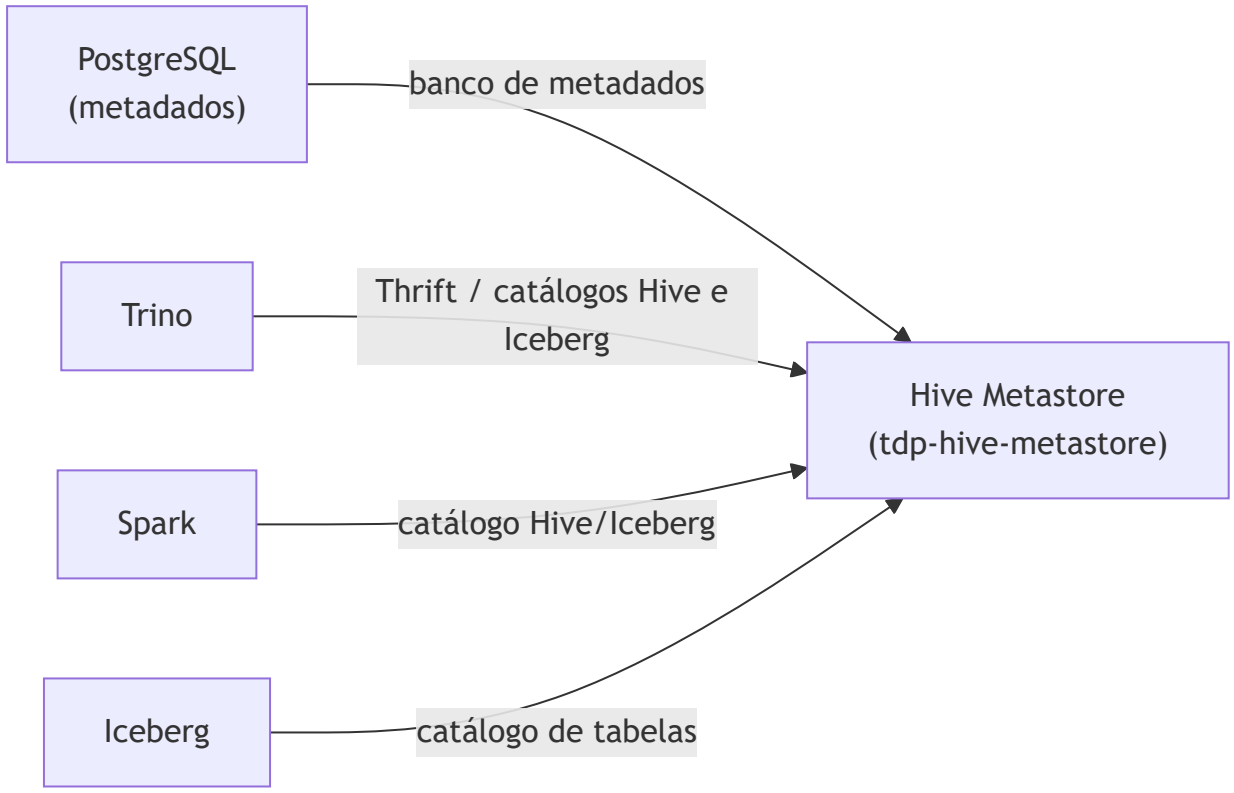
CHART

Version 3.0.1 | Type application | AppVersion 4.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O Hive Metastore funciona como repositório central de metadados para tabelas Hive e Iceberg no TDP. Outros componentes — principalmente Trino e Spark — conectam-se a ele via protocolo Thrift na porta `9083`.

O próprio Hive Metastore pode usar um PostgreSQL externo (`tdp-postgresql`) como seu banco de metadados — consulte Configuração do Hive Metastore.

Endereço do serviço

© Copyright Tecnisys 2026. All rights reserved.



O serviço do Hive Metastore dentro do cluster segue o padrão:

```
<RELEASE_NAME>-metastore.<NAMESPACE>.svc.cluster.local:9083
```

Exemplo com o release padrão `tdp-hive-metastore-project` no namespace `tdp-project`:

```
tdp-hive-metastore-project-metastore.tdp-project.svc.cluster.local:9083
```

Para verificar que o serviço tem endpoints ativos após o deploy:

```
kubectl get endpoints -n tdp-project tdp-hive-metastore-project-metastore
```

O campo `ENDPOINTS` deve mostrar o IP do pod na porta `9083`. Se exibir `<none>`, o seletor do serviço não corresponde ao pod — verifique se o deploy foi concluído e se os labels do pod correspondem ao seletor do Service.

Integração com Trino

O Trino usa o Hive Metastore como catálogo para tabelas Hive e Iceberg. A URI do Metastore é configurada na definição do catálogo Trino:

```
trino:
  additionalCatalogs:
    hive: |
      connector.name=hive
      hive.metastore.uri=thrift://tdp-hive-metastore-project-metastore.
<NAMESPACE>.svc.cluster.local:9083

    iceberg: |
      connector.name=iceberg
      hive.metastore.uri=thrift://tdp-hive-metastore-project-metastore.
<NAMESPACE>.svc.cluster.local:9083
```

Ajuste o endereço conforme o nome do release e o namespace utilizados no seu ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Integração com Spark (catálogo Iceberg)

Para jobs Spark que acessam tabelas Iceberg via Hive Metastore como catálogo:

```
maintenance:
  spark:
    config:
      "spark.sql.catalog.iceberg": "org.apache.iceberg.spark.SparkCatalog"
      "spark.sql.catalog.iceberg.type": "hive"
      "spark.sql.catalog.iceberg.uri": "thrift://tdp-hive-metastore-project-
metastore.<NAMESPACE>.svc.cluster.local:9083"
```

Para o chart `tdp-spark`:

```
spark:
  sparkConf:
    "spark.sql.catalog.hive_metastore":
"org.apache.spark.sql.hive.HiveExternalCatalog"
    "spark.hadoop.hive.metastore.uris": "thrift://tdp-hive-metastore-project-
metastore.<NAMESPACE>.svc.cluster.local:9083"
```

Verificar conectividade

Para testar se o Metastore está acessível a partir de outro pod no cluster:

```
kubectl -n <NAMESPACE> port-forward svc/tdp-hive-metastore-project-metastore
9083:9083
```

Com o port-forward ativo, ferramentas que suportam o protocolo Thrift podem conectar-se em `localhost:9083`.

Solução de problemas

Problema	Causa provável	Solução
Trino não encontra tabelas	URI do Metastore incorreta no catálogo	Verificar nome do serviço e namespace

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
Endpoints do serviço `<none>`	Seletor do Service não corresponde ao pod	Verificar se o deploy foi concluído e se os labels do pod correspondem ao seletor do Service
Timeout na conexão Thrift	Metastore sobrecarregado ou pod não pronto	Verificar `max_connections` do PostgreSQL e status do pod
Spark não acessa tabelas Hive	`hive.metastore.uris` não configurado	Verificar `spark.hadoop.hive.metastore.uris` nos valores Spark

Para a lista completa de parâmetros, use `helm show values` na versão do chart que instalou.

© Copyright Tecnisys 2026. All rights reserved.

Iceberg

CHART

Version 3.0.1 | Type application | AppVersion 1.10.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado

O que é o Apache Iceberg e por que ele precisa de manutenção?

O Apache Iceberg é um formato de tabela aberta para grandes datasets, projetado para superar limitações do Hive. Assim como o Delta Lake, o Iceberg mantém **snapshots**: versões imutáveis da tabela a cada operação de escrita. Com o uso contínuo, isso tende a acumular:

- **snapshots antigos**;
- **arquivos órfãos**;
- **arquivos pequenos** que prejudicam a performance de leitura.

O chart `tdp-iceberg` cria **CronJobs Kubernetes** para executar essas rotinas de manutenção de forma agendada, usando Apache Spark como motor de processamento.

> **PARA SABER MAIS**
>
> Consulte Apache Iceberg — Conceitos para uma visão completa do formato, snapshots e casos de uso.

Estrutura de valores (Helm)

O chart `tdp-iceberg` distribui a configuração em dois blocos principais:

© Copyright Tecnisys 2026. All rights reserved.



- `maintenance:` — habilita os jobs, define o Spark dos CronJobs, configurações Spark (catálogo, S3) e agendamentos de cada job.
- `spark:` — controla o subchart upstream Spark (pods master/worker). **Independente** da imagem usada pelos CronJobs.

```
maintenance:
enabled: true
spark:
  enabled: true
  image:
    repository: "registry.tecnisys.com.br/tdp-dev/images/spark"
    tag: "4.0.2-0"
jobs:
  expireSnapshots:
    enabled: true

spark:
image:
  registry: registry.tecnisys.com.br
  repository: tdp-dev/images/spark
  tag: 4.0.2-0
```

Visão geral

Propriedade	Valor
Chart	`tdp-iceberg`
Versão do Iceberg (runtime)	1.10.0
Versão do chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg`
Tipo	`maintenance` — CronJobs Kubernetes
Exposição HTTP externa	Não se aplica

© Copyright Tecnisys 2026. All rights reserved.



Compatibilidade de runtime

Componente	Versão
Spark	4.0.2
Iceberg (Spark runtime)	1.10.0
Scala	2.13

O chart não substitui o catálogo nem o warehouse: ele agenda **tarefas** que atuam sobre tabelas Iceberg já existentes. Você continua responsável por garantir que os comandos nos CronJobs apontem para o catálogo, metastore e bucket corretos.

Páginas relacionadas

- Integrações — Iceberg: S3, Hive Metastore, configuração do catálogo Spark, Trino, Airflow.
- Segurança — Iceberg: Secret `s3-credentials`, credenciais S3 e boas práticas.

Pré-requisitos

- **Kubernetes 1.32+**, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- **Helm 3.2.0+**
- Registry OCI da Tecnisys acessível pelo ambiente de instalação
- Secret `s3-credentials` criado no namespace antes do deploy (ver Segurança — Iceberg)
- Endpoint S3/MinIO acessível a partir do cluster
- Hive Metastore acessível via Thrift, se o catálogo Iceberg usar `type: hive`

Sem esses itens, os comandos Spark dos CronJobs falham ao resolver warehouse, credenciais ou metadados. Confirme que o metastore e o bucket referenciados nos valores apontam para o catálogo real das tabelas a serem mantidas.

Instalação

Instalação

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg \
  --version <CHART_VERSION> \
  -n <NAMESPACE> --create-namespace
```

Placeholder	Descrição
`<RELEASE_NAME>`	Nome do release Helm
`<NAMESPACE>`	Namespace Kubernetes de instalação
`<CHART_VERSION>`	Versão do chart

OpenShift

O chart configura `adaptSecurityContext: force` por padrão, adequando os contextos de segurança aos requisitos do OpenShift automaticamente. Nenhum parâmetro adicional é necessário para a maioria dos ambientes OpenShift.

```
global:
  compatibility:
    openshift:
      adaptSecurityContext: force
```

Verificar instalação

Terminal input

```
kubectl -n <NAMESPACE> get cronjobs
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<JOB_NAME>
```

Parâmetros principais

Parâmetros principais

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`maintenance.enabled`	Habilitar jobs de manutenção	`true`
`maintenance.spark.enabled`	Habilitar dependência Spark	`true`
`maintenance.spark.image.repository`	Imagem dos CronJobs	registry.tecnisys.com.br/tdp-dev/images/spark
`maintenance.spark.image.tag`	Tag da imagem dos CronJobs	4.0.2-0
`maintenance.spark.resources.requests.cpu`	CPU request dos jobs	`1`
`maintenance.spark.resources.requests.memory`	Memória request dos jobs	`2Gi`
`maintenance.spark.resources.limits.cpu`	CPU limit dos jobs	`2`
`maintenance.spark.resources.limits.memory`	Memória limit dos jobs	`4Gi`
`maintenance.jobs.expireSnapshots.enabled`	Habilitar expire snapshots	`true`
`maintenance.jobs.expireSnapshots.schedule`	Cron do expire snapshots	`0 2 * * *`
`maintenance.jobs.expireSnapshots.retentionDays`	Dias de retenção	`7`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`maintenance.jobs.removeOrphanFiles.enabled`	Habilitar remoção de órfãos	`true`
`maintenance.jobs.removeOrphanFiles.schedule`	Cron do remove orphan files	`0 3 * * 0`
`maintenance.jobs.removeOrphanFiles.olderThanDays`	Idade mínima dos órfãos	`3`
`maintenance.jobs.rewriteDataFiles.enabled`	Habilitar reescrita de dados	`false`
`maintenance.jobs.rewriteDataFiles.schedule`	Cron do rewrite data files	`0 1 * * 6`
`global.compatibility.openshift.adaptSecurityContext`	Compatibilidade OpenShift	`force`

Detalhes de configuração

Jobs de manutenção

Cada job vira um CronJob no Kubernetes: horário, política de retenção e o comando Spark são declarados no values. Monitore execuções bem-sucedidas, duração e consumo de cluster — jobs longos podem competir com outras cargas Spark.

Os jobs são configurados sob `maintenance.jobs.*`.

> **NOTE**
>
> Os exemplos abaixo mostram o formato esperado do comando. Ajuste catálogo, endpoint, warehouse, tabela e janelas de retenção conforme o seu ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Expire Snapshots

Remove snapshots antigos para liberar espaço de armazenamento e de metadados:

```
maintenance:
jobs:
  expireSnapshots:
    enabled: true
    schedule: "0 2 * * *"
    retentionDays: 7
    command: |
      spark-sql \
        --packages org.apache.iceberg:iceberg-spark-runtime-
4.0_2.13:1.10.0,org.apache.hadoop:hadoop-aws:3.3.4 \
        --conf spark.sql.catalog.iceberg=org.apache.iceberg.spark.SparkCatalog \
        --conf spark.sql.catalog.iceberg.type=hive \
        --conf spark.sql.catalog.iceberg.uri=thrift://metastore.hive-
metastore.svc.cluster.local:9083 \
        --conf spark.sql.catalog.iceberg.warehouse=s3a://warehouse/hive \
        --conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
\
        -e "CALL iceberg.system.expire_snapshots(older_than => TIMESTAMP '$(date -d
days ago' '+%Y-%m-%d %H:%M:%S')');"
```

Remove Orphan Files

Remove arquivos órfãos que deixaram de ser referenciados por snapshots válidos:

```
maintenance:
jobs:
  removeOrphanFiles:
    enabled: true
    schedule: "0 3 * * 0"
    olderThanDays: 3
    command: |
      spark-sql \
        --packages org.apache.iceberg:iceberg-spark-runtime-
4.0_2.13:1.10.0,org.apache.hadoop:hadoop-aws:3.3.4 \
        --conf spark.sql.catalog.iceberg=org.apache.iceberg.spark.SparkCatalog \
        --conf spark.sql.catalog.iceberg.type=hive \
        --conf spark.sql.catalog.iceberg.uri=thrift://metastore.hive-
metastore.svc.cluster.local:9083 \
```

© Copyright Tecnisys 2026. All rights reserved.

```
        --conf spark.sql.catalog.iceberg.warehouse=s3a://warehouse/hive \
        --conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
\
        -e "CALL iceberg.system.remove_orphan_files(older_than => TIMESTAMP '$(date
'3 days ago' '+%Y-%m-%d %H:%M:%S')');"
```

Rewrite Data Files

Reescreve e compacta arquivos de dados para melhorar a performance de leitura. Fica **desabilitado por padrão** por ser mais intensivo em recursos.

Ao habilitar, você aceita jobs que leem e gravam dados em volume — maior uso de CPU, memória e I/O no storage, e janela de execução possivelmente longa em tabelas grandes. Ative quando houver degradação de consultas por fragmentação de arquivos, após ingestões muito granulares ou quando a política de manutenção previr compactação periódica.

```
maintenance:
jobs:
  rewriteDataFiles:
    enabled: false
    schedule: "0 1 * * 6"
    command: |
      spark-sql \
        --packages org.apache.iceberg:iceberg-spark-runtime-
4.0_2.13:1.10.0,org.apache.hadoop:hadoop-aws:3.3.4 \
        --conf spark.sql.catalog.iceberg=org.apache.iceberg.spark.SparkCatalog \
        --conf spark.sql.catalog.iceberg.type=hive \
        --conf spark.sql.catalog.iceberg.uri=thrift://metastore.hive-
metastore.svc.cluster.local:9083 \
        --conf spark.sql.catalog.iceberg.warehouse=s3a://warehouse/hive \
        --conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
\
        -e "CALL iceberg.system.rewrite_data_files(table => 'iceberg.default.
<TABLE_NAME>');"
```

Parâmetros dos jobs

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição
`enabled`	Habilitar ou desabilitar o CronJob
`schedule`	Expressão cron
`retentionDays`	Dias de retenção para `expireSnapshots`
`olderThanDays`	Idade mínima para `removeOrphanFiles`
`command`	Script shell executado pelo container

Configuração do Spark

Este chart usa **dois contextos de imagem distintos** para o Spark — não os confunda:

Chave	O que controla	Formato
`spark.image.*`	Pods master/worker do subchart Spark	`registry` + `repository` separados
`maintenance.spark.image.*`	Containers dos CronJobs de manutenção	`repository` com URL completa

Atualizar uma não atualiza a outra — ao fazer upgrade do Spark, alinhe também a tag em `maintenance.spark.image` para evitar incompatibilidade.

Subchart Spark (master/worker)

```
spark:
image:
  registry: registry.tecnisys.com.br
  repository: tdp-dev/images/spark
  tag: 4.0.2-0
  pullPolicy: IfNotPresent
```

Imagem dos containers de manutenção

© Copyright Tecnisys 2026. All rights reserved.

```
maintenance:
spark:
  enabled: true
  image:
    repository: "registry.tecnisys.com.br/tdp-dev/images/spark"
    tag: "4.0.2-0"
    pullPolicy: IfNotPresent
```

> **NOTE**
>
> Estas duas configurações são independentes. Alterar `spark.image.*` não altera a imagem usada pelos CronJobs de manutenção.

Integrações

Para S3/MinIO, Hive Metastore, configuração do catálogo e uso a partir de Spark, Airflow ou Trino, consulte Integrações — Iceberg.

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

O chart `tdp-iceberg` é do tipo `maintenance` e **não cria PVCs**. A desinstalação remove apenas os CronJobs, Jobs, ServiceAccount, RBAC e demais recursos do release — sem risco de perda de dados nos buckets S3/Ozone.

© Copyright Tecnisys 2026. All rights reserved.

Iceberg – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 1.10.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado

O chart `tdp-iceberg` executa jobs de manutenção de tabelas Iceberg (expiração de snapshots, remoção de arquivos órfãos, reescrita de arquivos). Esses jobs acessam o armazenamento S3/MinIO via Spark e requerem o Secret Kubernetes `s3-credentials` com as credenciais de acesso ao bucket.

Credenciais e Secrets

Crie o Secret no namespace antes do deploy:

```
kubectl -n <NAMESPACE> create secret generic s3-credentials \
  --from-literal=access-key="<ACCESS_KEY>" \
  --from-literal=secret-key="<SECRET_KEY>"
```

Chave	Descrição
`access-key`	Chave de acesso ao bucket S3/MinIO
`secret-key`	Chave secreta para autenticação S3

O nome do Secret deve ser exatamente `s3-credentials`. Os CronJobs do chart referenciam esse nome diretamente via `env.secretKeyRef`.

Configuração do catálogo Iceberg

© Copyright Tecnisys 2026. All rights reserved.



O acesso ao Hive Metastore (usado como catálogo Iceberg) e ao endpoint S3 são configurados em `maintenance.spark.config`:

```
maintenance:
  spark:
    config:
      "spark.sql.catalog.iceberg": "org.apache.iceberg.spark.SparkCatalog"
      "spark.sql.catalog.iceberg.type": "hive"
      "spark.sql.catalog.iceberg.uri": "thrift://<RELEASE_NAME>-metastore.
<NAMESPACE>.svc.cluster.local:9083"
      "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>:<S3_PORT>"
      "spark.hadoop.fs.s3a.path.style.access": "true"
```

Ajuste o endereço do Metastore (`spark.sql.catalog.iceberg.uri`) se o nome do release ou namespace do Hive Metastore diferirem dos valores padrão.

Boas práticas

Aspecto	Recomendação
Credenciais	Não versionar `access-key` e `secret-key` em repositório Git
Rotação	Recriar o Secret e aguardar o próximo ciclo dos CronJobs
Acesso mínimo	Credenciais com permissão apenas nos buckets das tabelas Iceberg

Solução de problemas

Problema	Causa provável	Solução
CronJob falha com erro 403 no S3	Secret ausente ou chaves com nome incorreto	Verificar Secret `s3-credentials` no namespace

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
Erro de conexão ao Metastore	URI do Hive Metastore incorreto	Verificar endereço e porta 9083 acessíveis
Tabela não encontrada no catálogo	Namespace do catálogo Iceberg incorreto na query	Verificar configuração de `spark.sql.catalog.iceberg`

Para a lista completa de parâmetros, use `helm show values` na versão do chart que instalou.

© Copyright Tecnisys 2026. All rights reserved.

Iceberg – Integração

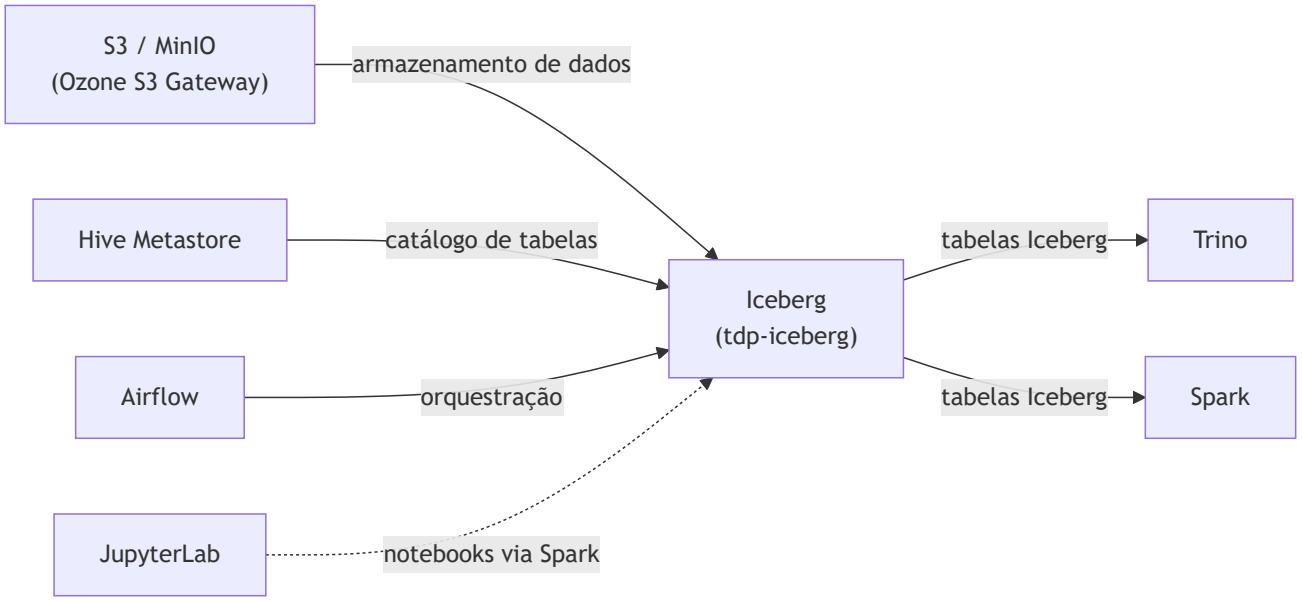
CHART

Version 3.0.1 | Type application | AppVersion 1.10.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O chart `tdp-iceberg` requer acesso ao armazenamento S3-compatível (Apache Ozone S3 Gateway, MinIO ou outro endpoint S3) e ao Hive Metastore como catálogo de tabelas.

Nesta página, o foco é mostrar **o que o ambiente precisa ter pronto** para que os jobs de manutenção funcionem. A ativação dos jobs, frequências e comandos fica em Configuração do Iceberg.

S3 / MinIO

Os CronJobs de manutenção requerem o Secret `s3-credentials` com as credenciais de acesso ao S3. Crie-o antes do deploy conforme descrito em Segurança — Iceberg.

© Copyright Tecnisys 2026. All rights reserved.

Hive Metastore

Por padrão, o catálogo Iceberg utiliza um Hive Metastore. Configure o URI de conexão via `maintenance.spark.config`:

```
maintenance:
  spark:
    config:
      "spark.sql.catalog.iceberg.type": "hive"
      "spark.sql.catalog.iceberg.uri": "thrift://<HIVE_METASTORE_SERVICE>.
<NAMESPACE>.svc.cluster.local:9083"
```

> **TIP**
>
> O valor típico é `thrift://metastore.hive-metastore.svc.cluster.local:9083`; ajuste *host* e *namespace* ao seu ambiente.

Configuração de conexões Spark

O exemplo completo com catálogo Iceberg + endpoint S3:

```
maintenance:
  spark:
    config:
      "spark.sql.catalog.iceberg": "org.apache.iceberg.spark.SparkCatalog"
      "spark.sql.catalog.iceberg.type": "hive"
      "spark.sql.catalog.iceberg.uri": "thrift://<HIVE_METASTORE_SERVICE>.
<NAMESPACE>.svc.cluster.local:9083"
      "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>.
<NAMESPACE>.svc.cluster.local:9000"
      "spark.hadoop.fs.s3a.path.style.access": "true"
```

Parâmetros de conexão

Parâmetro	Descrição	Exemplo
`spark.sql.catalog.iceberg`	Classe do catálogo Iceberg para Spark	`org.apache.iceberg.spark.SparkCatalog`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Exemplo
`spark.sql.catalog.iceberg.type`	Tipo do catálogo	`hive`
`spark.sql.catalog.iceberg.uri`	URI do Hive Metastore	`thrift://metastore.<NAMESPACE>.svc.cluster.local:9083`
`spark.hadoop.fs.s3a.endpoint`	URL do endpoint S3	`http://ozone-s3g.<NAMESPACE>.svc.cluster.local:9000`
`spark.hadoop.fs.s3a.path.style.access`	Forçar path-style (necessário para MinIO/Ozone)	`"true"`

Trino

O Trino pode consultar tabelas Iceberg via conector Iceberg. A configuração é realizada do lado do chart `tdp-trino` — consulte Configuração do Trino.

Spark

As tabelas Iceberg podem ser processadas diretamente pelo Spark. A configuração é realizada do lado do chart `tdp-spark` — consulte Configuração do Spark.

JupyterLab

O JupyterLab pode consultar tabelas Iceberg via Spark, desde que:

- a integração Jupyter ↔ Spark esteja funcional;
- o catálogo Iceberg esteja configurado no Spark;
- o Hive Metastore e o endpoint S3/MinIO estejam acessíveis.

Consulte:

- Integrações — JupyterLab
- Integrações — Spark

© Copyright Tecnisys 2026. All rights reserved.

Airflow

O Airflow pode orquestrar pipelines que operam tabelas Iceberg. A configuração de conexão é realizada do lado do chart `tdp-airflow` — consulte Integrações — Airflow.

Combinando arquivos de valores

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg \
  -n <NAMESPACE> \
  -f meu-values.yaml \
  -f values-integracao.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

JupyterLab

CHART

Version 3.0.1 | Type application | AppVersion 5.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | External DB suportado | Ingress suportado | Gateway API suportado

O que é o JupyterLab?

O **JupyterLab** é a interface interativa de notebooks do TDP Kubernetes. É o ambiente onde analistas e engenheiros de dados escrevem e executam código, organizam arquivos e exploram dados diretamente no browser.

No TDP Kubernetes, o JupyterLab é servido pelo **JupyterHub**, responsável pela autenticação e pelo ciclo de vida dos ambientes de cada usuário no cluster.

> **PARA SABER MAIS**
>
> Consulte JupyterLab — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

O chart `tdp-jupyter` organiza sua configuração em quatro blocos principais:

Bloco	Finalidade
`tdp-jupyter:`	Configuração do JupyterHub: Hub, Proxy, pods single-user, autenticação,

© Copyright Tecnisys 2026. All rights reserved.

Bloco	Finalidade
	storage e segurança
`tdpSparkIntegration:`	Integração com Spark: modo local ou cluster externo, ConfigMap e subchart opcional
`TDP-Settings:`	Configurações TDP: banco de dados externo e controle de exposição (Ingress/Gateway API)
`openshift:`	Forçar modo OpenShift quando a detecção automática não é possível

Use `helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter > values-padrao.yaml` para consultar todos os parâmetros disponíveis.

Visão geral

Propriedade	Valor
Chart	`tdp-jupyter`
Versão do JupyterHub	5.3.0
Versão do chart	3.0.1
Interface de uso	JupyterLab servido pelo JupyterHub
Modelo de execução	Um pod single-user por usuário autenticado
Banco de metadados	SQLite por padrão, com opção de PostgreSQL externo

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Exposição HTTP	Proxy do JupyterHub via Service, Ingress ou Gateway API

Componentes principais

Componente	Descrição
Hub	Serviço central do JupyterHub, responsável por autenticação e gestão dos ambientes
Proxy	Proxy reverso que encaminha cada usuário para o seu ambiente
Single-user pods	Um pod Kubernetes por usuário autenticado
PVCs	Armazenamento persistente para os notebooks, quando configurado

Modelo single-user

Cada vez que um usuário faz login, o Hub cria um pod dedicado para ele no Kubernetes. Quando o usuário para o servidor, o pod é encerrado, mas o storage pode persistir se estiver associado a PVC.

Isso significa que o dimensionamento do cluster deve considerar o pico esperado de usuários simultâneos.

Kernel padrão (Python 3.12)

Ao iniciar, o pod single-user executa um `postStart` hook que torna o kernel `py312` (Python 3.12, com as bibliotecas Spark/Delta/Iceberg pré-instaladas) o kernel padrão exibido como **"Python 3"** no notebook — o kernel `python3` original é substituído pela

© Copyright Tecnisys 2026. All rights reserved.

cópia do `py312`. A configuração `KernelSpecManager.allowed_kernelspecs` restringe a lista de kernels disponíveis ao usuário a esse kernel.

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- Helm 3.2.0+
- Provisionador de PV disponível no cluster
- Cluster Spark disponível via `tdp-spark` ou instalação compatível, quando for usar a integração Spark

Páginas relacionadas

- Integrações — JupyterLab
- Segurança — JupyterLab
- Exposição externa

Instalação

Instalação (OCI)

Kubernetes

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter \
  -n <NAMESPACE> --create-namespace --timeout 10m
```

Sem um arquivo `-f`, utilizam-se os valores padrão do chart. Em ambientes reais, versione um arquivo de valores com recursos, storage e tipo de Service alinhados à política do cluster.

OpenShift

O chart detecta automaticamente o ambiente OpenShift por meio da API de descoberta do Kubernetes. Se a detecção automática não for possível (redes restritas, pipelines CI), force o modo OpenShift explicitamente:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter \
  -n <NAMESPACE> --create-namespace --timeout 10m \
  --set openshift.enabled=true \
  --set tdp-
jupyter.scheduling.userScheduler.containerSecurityContext.runAsNonRoot=true
```

Ou use um arquivo overlay `values-openshift.yaml`:

```
openshift:
  enabled: true

tdp-jupyter:
  scheduling:
    userScheduler:
      containerSecurityContext:
        runAsNonRoot: true
```

OPENSHIFT — O QUE MUDA

Quando o modo OpenShift está ativo, o chart aplica automaticamente:

- Volumes de overlay para `/etc/passwd` e `/etc/group` nos pods single-user
- Init container `openshift-passwd-bootstrap` para mapear UID/GID
- Hook pós-install que nulifica `runAsUser/runAsGroup` no Hub e força `runAsNonRoot: true`

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão
`tdp-jupyter.enabled`	Habilitar JupyterHub	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-jupyter.hub.resources.requests.cpu`	CPU mínima do Hub	`500m`
`tdp-jupyter.hub.resources.requests.memory`	Memória mínima do Hub	`1Gi`
`tdp-jupyter.hub.resources.limits.cpu`	CPU máxima do Hub	`1000m`
`tdp-jupyter.hub.resources.limits.memory`	Memória máxima do Hub	`2Gi`
`tdp-jupyter.hub.config.JupyterHub.authenticator_class`	Classe de autenticação	`dummy`
`tdp-jupyter.hub.config.DummyAuthenticator.username`	Usuário padrão (autenticação básica)	`admin`
`tdp-jupyter.hub.config.DummyAuthenticator.password`	Senha padrão (alterar em produção)	`tdp-password`
`tdp-jupyter.singleuser.cpu.limit`	CPU máxima por notebook	`4`
`tdp-jupyter.singleuser.cpu.guarantee`	CPU garantida por notebook	`2`
`tdp-jupyter.singleuser.memory.limit`	Memória máxima por notebook	`8G`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-jupyter.singleuser.memory.guarantee`	Memória garantida por notebook	`4G`
`tdp-jupyter.singleuser.storage.capacity`	Tamanho do PVC por usuário	`5Gi`
`tdp-jupyter.singleuser.storage.dynamic.storageClass`	StorageClass do PVC de usuário	`null` (padrão do cluster)
`tdp-jupyter.proxy.service.type`	Tipo de Service do Proxy	`NodePort`
`tdp-jupyter.proxy.service.nodePorts.http`	Porta HTTP NodePort	`30080`
`tdp-jupyter.proxy.service.nodePorts.https`	Porta HTTPS NodePort	`30443`
`TDP-Settings.externalDatabase.enabled`	Habilitar banco de dados externo	`false`
`TDP-Settings.externalDatabase.recreate`	Recriar banco ao atualizar	`false`
`TDP-Settings.gateway.ingress.enabled`	Habilitar Ingress	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilitar Gateway API	`false`
`tdpSparkIntegration.enabled`	Habilitar integração com Spark	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdpSparkIntegration.deployS parkCluster`	Instalar subchart `tdp-spark` junto	`false`
`openshift.enabled`	Forçar modo OpenShift	`false`

Detalhes de configuração

Banco externo (PostgreSQL)

Por padrão, o Hub usa SQLite com armazenamento em PVC. Para ambientes de produção com múltiplas réplicas ou quando a persistência do Hub precisa ser gerenciada fora do ciclo de vida do release Helm, use PostgreSQL externo.

```
TDP-Settings:
  externalDatabase:
    enabled: true
    recreate: false
    externalSecret:
      releaseName: "<POSTGRESQL_RELEASE>"

tdp-jupyter:
  hub:
    db:
      type: postgres
      url: "postgresql://<DATABASE_USER>@<POSTGRESQL_SERVICE>.
<NAMESPACE>.svc.cluster.local:5432/<DATABASE_NAME>"
      password: null
      upgrade: false
    extraEnv:
      PGPASSWORD:
        valueFrom:
          secretKeyRef:
            name: "<SECRET_NAME>"
            key: password
```

Storage

© Copyright Tecnisys 2026. All rights reserved.

Cada usuário autenticado pode receber um volume persistente para o diretório de trabalho. A StorageClass define onde e como o disco é provisionado.

```
singleuser:
  storage:
    dynamic:
      storageClass: <STORAGE_CLASS>
    capacity: 5Gi
```

Escolha a `storageClass` compatível com o provisionador disponível no cluster e defina a `capacity` de acordo com o uso esperado de dados locais nos notebooks.

Host aliases

Quando os pods de notebook precisam resolver hostnames que não estão no DNS do cluster — por exemplo, para acessar um endpoint de armazenamento S3 interno —, use `singleuser.extraPodConfig.hostAliases`:

```
tdp-jupyter:
  singleuser:
    extraPodConfig:
      hostAliases:
        - ip: "<CONTROLLER_IP>"
          hostnames:
            - "<COMPONENT_HOSTNAME>"
```

Esse parâmetro adiciona entradas ao arquivo `/etc/hosts` de cada pod de notebook, permitindo que o Spark e as bibliotecas Python resolvam o hostname para o IP informado.

Integração com Spark

O JupyterLab pode funcionar sem integração com Spark (modo `local[*]`) ou ligar-se a um cluster Spark externo. Os detalhes completos, modos de operação e troubleshooting ficam em Integrações — JupyterLab.

Acesso & Segurança

© Copyright Tecnisys 2026. All rights reserved.

Exposição do serviço

A forma de expor o Proxy define como os usuários chegam à interface.

NodePort (padrão)

```
tdp-jupyter:
  proxy:
    service:
      type: NodePort
      nodePorts:
        http: 30080
        https: 30443
```

Terminal input

```
kubectl get nodes -o wide
# Acesse via http://<NODE_IP>:30080
```

Port-forward

Terminal input

```
kubectl -n <NAMESPACE> port-forward service/<RELEASE_NAME>-proxy 8080:80
# Acesse via http://localhost:8080
```

LoadBalancer

```
tdp-jupyter:
  proxy:
    service:
      type: LoadBalancer
```

Terminal input

```
kubectl get svc <RELEASE_NAME>-proxy -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Use o IP externo fornecido pelo provisionador de LoadBalancer para acessar a interface.

ClusterIP

```
tdp-jupyter:
  proxy:
    service:
      type: ClusterIP
```

Restringe o acesso ao interior do cluster (apenas interno).

Exposição externa

Para exposição externa estável por Ingress ou Gateway API, consulte a página de exposição externa.

Se o ambiente exigir exposição direta pelo Service, `tdp-jupyter.proxy.service.type` pode ser ajustado para `LoadBalancer` no arquivo de valores. Essa é uma decisão de Service Kubernetes, não um modo de exposição HTTP/HTTPS equivalente aos modelos Ingress/Gateway API.

Autenticação

A configuração detalhada de autenticação, LDAP e boas práticas de acesso fica em Segurança — JupyterLab.

Solução de problemas

Solução de problemas

Terminal input

```
# Status dos pods
kubectl -n <NAMESPACE> get pods -l app.kubernetes.io/instance=<RELEASE_NAME>

# Logs do Hub
kubectl -n <NAMESPACE> logs -l app.kubernetes.io/component=hub

# Logs do Proxy
kubectl -n <NAMESPACE> logs -l app.kubernetes.io/component=proxy
```

© Copyright Tecnisys 2026. All rights reserved.

```
# Eventos de pod
kubectl -n <NAMESPACE> describe pod <POD_NAME>

# PVCs
kubectl -n <NAMESPACE> get pvc -l app.kubernetes.io/instance=<RELEASE_NAME>

# Banco externo (se configurado)
kubectl -n <NAMESPACE> exec -it <POD_NAME> -- \
  psql "postgresql://<DATABASE_USER>@<POSTGRESQL_SERVICE>.
<NAMESPACE>.svc.cluster.local:5432/<DATABASE_NAME>"
```

Sintoma	Causa provável	Ação sugerida
Pods em `Pending`	StorageClass inexistente ou sem capacidade	Verifique provisionador de PV e cotas
Falha de conexão Spark	NetworkPolicy ou serviço Spark inacessível	Verifique `tdpSparkIntegration.enabled` e conectividade
Falha de conexão com banco externo	PostgreSQL inacessível ou credenciais incorretas	Verifique URL, Secret e acessibilidade do PostgreSQL
Pods OOMKilled	Limites de CPU/memória muito baixos	Ajuste `hub.resources` e `singleuser.cpu`/`singleuser.memory`

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
kubectl delete configmap <RELEASE_NAME>-spark-integration -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

PVCS

Os PVCs dos usuários **não são removidos** automaticamente. Para removê-los:

Terminal input

```
kubectl delete pvc -l app.kubernetes.io/instance=<RELEASE_NAME> -n
<NAMESPACE>
```

Essa operação é **destrutiva** e apaga os dados de notebook de todos os usuários.

© Copyright Tecnisys 2026. All rights reserved.

JupyterLab – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 5.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado

O chart `tdp-jupyter` usa por padrão autenticação **`DummyAuthenticator`** (usuário e senha definidos nos valores) e permite **LDAP** opcional via `ldapauthenticator.LDAPAuthenticator`.

Autenticação padrão

```
tdp-jupyter:
  hub:
    config:
      JupyterHub:
        authenticator_class: dummy
      DummyAuthenticator:
        username: admin
        password: tdp-password
```

> **WARNING**
>
> Adequado apenas para desenvolvimento. Altere a senha ou o método de autenticação em produção.

LDAP

A autenticação LDAP é opcional. Para habilitá-la, crie o Secret com a senha de bind e aplique a configuração de overlay.

© Copyright Tecnisys 2026. All rights reserved.

Credenciais e Secrets

Terminal input

```
kubectl create secret generic tdp-jupyter-ldap-bind \
  --from-literal=LDAP_BIND_PASSWORD='<LDAP_BIND_PASSWORD>' \
  -n <NAMESPACE>
```

Configuração LDAP

```
tdp-jupyter:
  hub:
    config:
      JupyterHub:
        authenticator_class: ldapauthenticator.LDAPAuthenticator
      LDAPAuthenticator:
        server_address: "<LDAP_HOST>"
        server_port: 389
        tls_strategy: "insecure"
        user_attribute: "uid"
        bind_dn_template:
          - "uid={username},cn=users,cn=accounts,dc=<LDAP_DOMAIN>,dc=<TLD>"
    extraConfig:
      ldap-config: |
        c.Authenticator.allow_all = True
```

O bloco `allow_all` é **opcional** e serve para facilitar testes. Em produção, restrinja com as políticas suportadas pelo `LDAPAuthenticator` da sua versão do JupyterHub (consulte a documentação oficial para `allowed_groups` e listas de usuários).

Parâmetros

Parâmetro	Descrição
`LDAPAuthenticator.server_address`	Endereço do servidor LDAP
`LDAPAuthenticator.server_port`	Porta do servidor LDAP

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição
`tls_strategy`	Modo TLS (ex.: `insecure` para desenvolvimento)
`bind_dn_template`	Template do DN do usuário
`user_attribute`	Atributo de login (ex.: `uid`)

NetworkPolicy do servidor single-user

O chart aplica uma NetworkPolicy aos pods single-user que restringe tanto **ingress** quanto **egress**.

Egress

O bloco `singleuser.networkPolicy` controla o tráfego de saída (egress) permitido para os pods de notebook do usuário. Por padrão (`enabled: true`), as regras de egress liberam:

```
tdp-jupyter:
  singleuser:
    networkPolicy:
      enabled: true
      egress:
        # Resolução DNS
        - to:
          - namespaceSelector:
              matchLabels:
                kubernetes.io/metadata.name: kube-system
          ports:
          - protocol: UDP
            port: 53
          - protocol: TCP
            port: 53
        # Comunicação com o Hub (mesmo namespace)
        - to:
          - namespaceSelector: {}
        # Acesso a IPs externos ao cluster (ex.: Ozone S3, serviços externos)
        - to:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    - ipBlock:
        cidr: 0.0.0.0/0
  # Spark Master no mesmo namespace
  - to:
    - podSelector:
        matchLabels:
          app.kubernetes.io/component: master
          app.kubernetes.io/name: spark
    ports:
    - protocol: TCP
      port: 7077
```

Regra	Finalidade
DNS (`kube-system`, portas 53 UDP/TCP)	Resolução de nomes
`namespaceSelector: {}`	Comunicação com o Hub e demais pods do namespace
`ipBlock: 0.0.0.0/0`	Acesso a serviços externos ao cluster (Ozone S3G, Trino, Kafka, Hive Metastore, etc.)
`podSelector` Spark master (porta 7077)	Conexão com o cluster Spark quando `tdpSparkIntegration.enabled: true`

Ajuste ou restrinja `egress` conforme a política de rede do ambiente — por exemplo, substituindo `0.0.0.0/0` por `ipBlock.cidr` mais específicos para os serviços S3/Ozone, Trino ou Hive Metastore usados pelo notebook.

Ingress (Workers Spark → driver)

Em modo de cluster distribuído, o driver Spark roda **dentro do pod do notebook** — são os Workers que precisam abrir conexão de volta para ele, não o contrário.
Por isso é necessário permitir **ingress** dos pods Worker nas portas do driver (`2222`) e do block manager (`7777`) do notebook, para que os Workers consigam entregar os resultados ao driver:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-jupyter:
  singleuser:
    networkPolicy:
      enabled: true
      # Permite que os Workers Spark se conectem de volta ao driver do notebook
      ingress:
        - from:
            - podSelector:
                matchLabels:
                  app.kubernetes.io/component: worker
                  app.kubernetes.io/instance: tdp-spark
                  app.kubernetes.io/name: spark
          ports:
            - protocol: TCP
              port: 2222   # spark.driver.port
            - protocol: TCP
              port: 7777   # spark.blockManager.port
```

> **WARNING**
>
> Sem essa regra de ingress, os Workers entram em timeout ao tentar alcançar o driver e a aplicação Spark falha com `Connecting to /<pod-ip>:2222 timed out`.

Solução de problemas

Problema	Solução
Falha de login LDAP	Verifique `bind_dn_template`, conectividade e atributos do servidor LDAP
Hub não inicia	Valide a sintaxe de `extraConfig` e o YAML
Falha de conexão Spark/S3/Trino a partir do notebook	Verifique `singleuser.networkPolicy.egress` — tráfego pode estar sendo bloqueado

© Copyright Tecnisys 2026. All rights reserved.

Problema	Solução
Workers não conseguem alcançar o driver (`Connecting to /<ip>:2222 timed out`)	A NetworkPolicy do pod single-user está bloqueando o ingress dos pods Worker do Spark. Adicione a regra de `ingress` da seção Ingress (Workers Spark → driver) e atualize o release

© Copyright Tecnisys 2026. All rights reserved.

JupyterLab – Integração

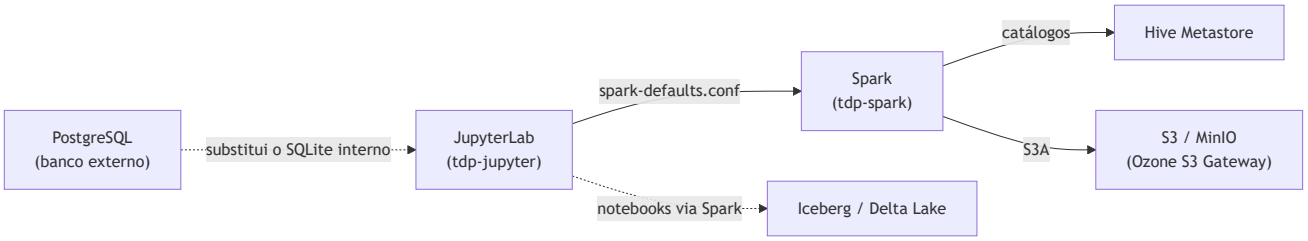
CHART

Version 3.0.1 | Type application | AppVersion 5.3.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O JupyterHub usa SQLite internamente por padrão; um PostgreSQL externo (`tdp-postgresql`) pode substituí-lo — consulte Configuração do Jupyter.

Integração com Spark

O chart `tdp-jupyter` integra-se com o Apache Spark via o mecanismo `tdpSparkIntegration`. Ao ativar a integração, um ConfigMap (`tdp-jupyter-spark-integration`) é criado com as configurações de `spark-defaults.conf` e um script auxiliar `jupyter-spark-env.sh`.

Os pods de notebooks montam esse ConfigMap em `/opt/bitnami/spark/conf` e executam o script no `postStart`, de modo que cada sessão Spark encontra o master correto automaticamente.

Para o usuário final, a decisão principal é simples:

- usar **PySpark local** para testes rápidos e desenvolvimento;
- usar **cluster Spark externo** quando quiser distribuir processamento;
- usar **Iceberg a partir do notebook** apenas depois de a integração Spark e o catálogo Iceberg já estarem configurados no ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Modos de operação

Modo	`tdpSparkIntegration.enabled`	Valor resolvido para `spark.master`	Uso típico
PySpark local	`false`	`local[*]`	Executa Spark dentro do próprio pod do notebook (padrão para desenvolvimento)
Cluster externo	`true`	`spark://<RELEASE_NAME>-spark-master-svc.<NAMESPACE>.svc.cluster.local:7077`	Liga a um deployment Spark já existente

> **TIP**
>
> O parâmetro `spark.master` no `values.yaml` fica vazio por padrão. O template escolhe o valor correto em tempo de renderização com base em `tdpSparkIntegration.enabled`. É possível fornecer uma URL personalizada se necessário.

Componentes envolvidos

Componente	Função
`templates/spark-integration-configmap.yaml`	Renderiza as configurações Spark e o script auxiliar de ambiente
`singleuser.extraEnv`	Define variáveis de ambiente Spark para cada pod de notebook

© Copyright Tecnisys 2026. All rights reserved.

Componente	Função
`singleuser.lifecycleHooks.postStart`	Executa `jupyter-spark-env.sh` antes do JupyterLab iniciar
`singleuser.networkPolicy.egress`	Permite que os pods de notebook alcancem o Spark master e serviços auxiliares

Variáveis de ambiente injetadas nos pods de notebook

```
SPARK_HOME=/opt/bitnami/spark
PYTHONPATH=/opt/bitnami/spark/python:/opt/bitnami/spark/python/lib/py4j-
0.10.9.7-src.zip
SPARK_CONF_DIR=/opt/bitnami/spark/conf
PYSPARK_PYTHON=/opt/conda/envs/py312/bin/python
PYSPARK_DRIVER_PYTHON=/opt/conda/envs/py312/bin/python
SPARK_MASTER_URL=<automático>  # local[*] ou spark://... conforme
tdpSparkIntegration.enabled
SPARK_DRIVER_PORT=2222
SPARK_BLOCKMANAGER_PORT=7777
```

Volumes montados nos pods de notebook

Caminho	Tipo	Conteúdo
`/opt/bitnami/spark/conf`	ConfigMap	`spark-defaults.conf` e scripts auxiliares
`/tmp/spark-local`	emptyDir	Dados temporários e shuffle do Spark
`/tmp/spark-logs`	emptyDir	Logs do driver Spark

Como configurar

Modo 1 — PySpark local (padrão)

© Copyright Tecnisys 2026. All rights reserved.

Não requer um cluster Spark externo. O Spark é executado dentro do pod do notebook com `local[*]`:

```
tdpSparkIntegration:
  enabled: false
  deploySparkCluster: false
  configMap:
    sparkConfig:
      "spark.master": ""  # resolve para local[*]
```

Teste num notebook:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Local PySpark").getOrCreate()
print(spark.sparkContext.master)  # local[*]
```

Modo 2 — Cluster Spark externo

Liga os notebooks a um deployment Spark já existente no cluster Kubernetes:

```
tdpSparkIntegration:
  enabled: true
  deploySparkCluster: false   # false = apontar para deployment existente
  configMap:
    sparkConfig:
      "spark.kubernetes.namespace": "<NAMESPACE>"   # opcional
      "spark.master": ""       # resolve para spark://<RELEASE_NAME>-spark-
master-svc.<NAMESPACE>:7077
      "spark.driver.host": ""  # deixe vazio para usar o serviço admin do
notebook
      "spark.executor.instances": "2"
      "spark.executor.memory": "4g"
      "spark.executor.cores": "3"

tdp-spark:
  spark:
    worker:
      replicaCount: 2
      resources:
```

© Copyright Tecnisys 2026. All rights reserved.

```
        limits:
          cpu: 4
          memory: 6Gi
```

Certifique-se de que o serviço Spark master é acessível a partir do namespace dos notebooks (ex.: `tdp-spark-master-svc.tdp-project.svc.cluster.local:7077`).

Considerações sobre NetworkPolicy

- Os pods de notebook adicionam uma regra de egress que corresponde a qualquer Spark master (`app.kubernetes.io/component: master`, `app.kubernetes.io/name: spark`).
- Se o chart Spark tiver a sua própria NetworkPolicy, permita conexões de entrada a partir do namespace dos notebooks.
- Para que os Workers Spark consigam se conectar de volta ao driver do notebook, configure também a regra de ingress descrita em Segurança — JupyterLab.

Limitações conhecidas

Cada notebook pode iniciar uma sessão Spark própria, com portas fixas pré-configuradas para o driver (`2222`) e para o BlockManager (`7777`).
Se múltiplas sessões Spark forem abertas no mesmo pod — por exemplo, vários kernels ativos — ou se uma sessão anterior não liberou os recursos corretamente, o Spark pode encontrar essas portas já em uso e gerar `BindException`.
Quando o ambiente usa Spark Connect, definir `SPARK_CONNECT_PORT: "0"` faz o endpoint escolher uma porta livre aleatória, evitando conflito especificamente na porta padrão `15002`; essa configuração não substitui o tratamento das portas do driver e do BlockManager.

```
tdp-jupyter:
  singleuser:
    extraEnv:
      SPARK_CONNECT_PORT: "0"   # porta aleatória, evita conflitos quando vários notebooks rodam ao mesmo tempo
```

Modo 3 — Cluster Spark integrado (opcional)

Define `tdpSparkIntegration.deploySparkCluster: true` para instalar o subchart `tdp-spark` junto com o JupyterHub:

© Copyright Tecnisys 2026. All rights reserved.

```
tdpSparkIntegration:
  enabled: true
  deploySparkCluster: true
```

Ajuste os valores do subchart `tdp-spark` conforme necessário.

Uso de Iceberg a partir do Jupyter

O suporte a Iceberg no Jupyter não é uma integração separada do chart `tdp-jupyter`. Na prática, ele acontece **via Spark**:

- o notebook liga-se ao Spark;
- o Spark precisa conhecer o catálogo Iceberg;
- o catálogo Iceberg precisa de acesso ao Hive Metastore e ao storage S3/MinIO.

Por isso:

- a configuração do Jupyter fica nesta página;
- a configuração do catálogo Iceberg fica em Integrações — Iceberg;
- a configuração do Spark fica em Integrações — Spark.

> **NOTE**
>
> Não trate Iceberg como obrigatório para o Jupyter. Ele é apenas um cenário adicional para notebooks que precisem consultar ou manter tabelas Iceberg via Spark.

Uso de Delta Lake a partir do Jupyter

O suporte a Delta Lake no Jupyter também não é uma integração separada do chart `tdp-jupyter` — assim como o Iceberg, acontece **via Spark**:

- o notebook liga-se ao Spark;
- o bloco `deltaLake` do chart `tdp-spark` habilita o suporte, mas não configura sozinho o `spark.sparkConf` — as propriedades Spark necessárias são fornecidas via `customSparkConfig.properties` ou `spark.sparkConf`;
- diferente do Iceberg, as tabelas Delta Lake **não passam pelo Hive Metastore** — o acesso é direto aos caminhos no storage S3/MinIO.

© Copyright Tecnisys 2026. All rights reserved.

Por isso:

- a configuração do Jupyter fica nesta página;
- a configuração do bloco Delta Lake fica em Integrações — Delta Lake;
- a configuração do Spark fica em Integrações — Spark.

Instalar ou atualizar o JupyterHub

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter \
  -n <NAMESPACE> \
  -f values.yaml
```

> **APÓS ATUALIZAÇÕES**
>
> Sempre que modificar ConfigMaps ou variáveis de ambiente, reinicie os pods de usuário (Stop Server → Start Server no JupyterHub) para que as novas configurações entrem em vigor.

Checklist de verificação

1. **Pods em execução**

   Terminal input

   ```
   kubectl get pods -n <NAMESPACE> | grep jupyter
   ```

2. **ConfigMap Spark criado**

   Terminal input

   ```
   kubectl get configmap tdp-jupyter-spark-integration -n <NAMESPACE> -o yaml
   ```

© Copyright Tecnisys 2026. All rights reserved.

3. **Conectividade de rede** (a partir de um pod de notebook)

Terminal input

```
kubectl exec -n <NAMESPACE> <POD_NAME> -- \
  curl -sv tdp-spark-master-svc.<NAMESPACE>.svc.cluster.local:7077
```

Testar a integração

Notebook de teste incluído no chart

O chart inclui um notebook de teste (`tdp-jupyter-spark-test` ConfigMap). Para o extrair:

Terminal input

```
kubectl get configmap tdp-jupyter-spark-test -n <NAMESPACE> \
  -o jsonpath='{.data.spark-integration-test\.ipynb}' \
  > spark-integration-test.ipynb
```

Faça upload do notebook pelo JupyterLab e execute cada célula.

Smoke test manual

Execute o seguinte código num notebook para validar a integração:

```
import os
from pyspark.sql import SparkSession

print("SPARK_HOME:", os.environ.get("SPARK_HOME"))
print("SPARK_MASTER_URL:", os.environ.get("SPARK_MASTER_URL"))

spark = SparkSession.builder.appName("TDP-Jupyter Smoke Test").getOrCreate()
print("Versão Spark:", spark.version)
print("Master activo:", spark.sparkContext.master)

spark.range(5).show()
spark.stop()
```

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Sintoma	Causa provável	Ação sugerida
`JAVA_GATEWAY_EXITED` ou erros Py4J	`SPARK_HOME` / `PYTHONPATH` mal configurados	Verifique se `singleuser.extraEnv` usa os caminhos `/opt/bitnami/spark`
`IllegalStateException: Cannot call methods on a stopped SparkContext`	Spark master inacessível ou NetworkPolicy bloqueando egress/ingress	Confirme `tdpSparkIntegration.enabled`, verifique o serviço Spark, ajuste as NetworkPolicies
Pod do notebook falha na inicialização (`ImportError` para `zmq`)	`PYTHONPATH` poluído com site-packages PySpark	Não adicione `/opt/conda/envs/py312/lib/python3.12/site-packages` ao `PYTHONPATH`
Driver Spark não consegue conectar/comunicar	`SPARK_DRIVER_HOST` não resolvível	Deixe em branco para usar o serviço admin do notebook ou forneça uma entrada DNS acessível
Workers não conseguem alcançar o driver (`Connecting to /<ip>:2222 timed out`)	A NetworkPolicy do pod single-user está bloqueando o ingress dos pods Worker do Spark	Adicione a regra de `ingress` descrita em Segurança — JupyterLab e atualize o release
`java.net.UnknownHostException: <pod-name>`	`spark.driver.host` está resolvendo para o hostname do pod em vez do IP	Garanta que `spark.driver.host` esteja vazio em `sparkConfig` e que `SPARK_DRIVER_HOST` seja injetado via Downward API (`fieldPath: status.podIP`)

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Causa provável	Ação sugerida
Avisos `CANNOT_MODIFY_CONFIG`	Configuração Spark aplicada via `SparkSession.builder.config()` após o import do PySpark	Passe `spark.driver.host` e JARs via `PYSPARK_SUBMIT_ARGS` antes de importar o PySpark, não via `SparkSession.builder.config()`

Comandos de diagnóstico

Terminal input

```
# Logs do pod do notebook
kubectl logs -n <NAMESPACE> <POD_NAME>

# Variáveis de ambiente Spark dentro do pod
kubectl exec -n <NAMESPACE> <POD_NAME> -- env | grep SPARK

# Listar arquivos montados
kubectl exec -n <NAMESPACE> <POD_NAME> -- ls -R /opt/bitnami/spark/conf

# Verificar endpoints do serviço Spark master
kubectl get svc -n <NAMESPACE> | grep spark-master
```

Personalização avançada

- Adicione propriedades Spark extra em `tdpSparkIntegration.configMap.sparkConfig`.
- Defina perfis de tamanho de notebook via `singleuser.profileList` e ajuste variáveis de ambiente Spark por perfil.
- Quando usar múltiplos clusters Spark, sobreponha `spark.master` por perfil ou via ambiente de usuário.

Limpeza

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl delete configmap tdp-jupyter-spark-integration -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Kafka

CHART

Version 3.0.1 | Type application | AppVersion 4.1.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | Ingress suportado | Gateway API suportado

O chart `tdp-kafka` implanta um cluster **Apache Kafka** gerenciado pelo operador **Strimzi**, além de uma interface web (**Kafka UI**) e suporte a conectores **Debezium** via Kafka Connect.

Visão geral

Propriedade	Valor
Chart	`tdp-kafka`
Kafka	4.1.0
Versão do Chart	3.0.1

Páginas relacionadas

- Integrações — Kafka — Kafka Connect, Debezium e integração com clientes
- Segurança — Kafka — usuários, autenticação, TLS e políticas
- Exposição externa: Ingress e Gateway API — exposição HTTP/HTTPS da Kafka UI

O que é o Strimzi?

O Strimzi é um operador Kubernetes especializado no ciclo de vida de clusters Kafka. Em vez de gerenciar processos Kafka manualmente, o Strimzi traduz recursos

© Copyright Tecnisys 2026. All rights reserved.

Kubernetes nativos (como `KafkaNodePool` e `Kafka`) em brokers configurados, monitorados e atualizados de forma declarativa.

A versão 4.1.0 do chart `tdp-kafka` opera no modo **KRaft** (Kafka sem ZooKeeper). No Kafka 4.x o modo ZooKeeper foi removido, então o gerenciamento de metadados é feito pelo próprio Kafka via KRaft, o que simplifica a topologia e reduz a latência de operações de metadados.

> **PARA SABER MAIS**
>
> Consulte Apache Kafka — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Componentes implantados

Componente	Descrição
Cluster Kafka	Brokers e controllers gerenciados pelo Strimzi via `KafkaNodePool`
Entity Operator	Gerencia `KafkaTopic` e `KafkaUser` como recursos Kubernetes
Kafka UI	Interface web para monitoramento, navegação de tópicos e mensagens
Kafka Connect	Infraestrutura para conectores Debezium (CDC) — opcional

Estrutura de valores (Helm)

O chart `tdp-kafka` distribui a configuração nos blocos abaixo:

- **Blocos raiz do cluster Kafka** — `name`, `kafkaVersion`, `metadataVersion`, `clusterLabel`, `listeners.*`, `security.*`, `config.*`, `entityOperator.*` e `nodePool.*`.
- `tdp-kafka:` — habilitação do chart.

© Copyright Tecnisys 2026. All rights reserved.



- `kafka-ui:` — configuração da interface web, autenticação, RBAC, conexão com o cluster e Service.
- `kafkaConnects:` — conectores Debezium opcionais para bancos como PostgreSQL, MySQL, SQL Server e Oracle.
- `registryPushSecret:` — Secret opcional para *push* de imagens de conectores.
- `TDP-Settings:`, `ingress:` e `gatewayApi:` — controle de exposição da Kafka UI por Ingress ou Gateway API.

```
name: tdp-kafka
kafkaVersion: 4.1.0
metadataVersion: 4.1-IV1

listeners:
plain:
  enabled: true
tls:
  enabled: true

nodePool:
name: brokers
replicas: 4

tdp-kafka:
enabled: true

kafka-ui:
enabled: true

kafkaConnects:
postgres:
  enabled: false

TDP-Settings:
gateway:
  ingress:
    enabled: false
  gatewayApi:
    enabled: false
```

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.

© Copyright Tecnisys 2026. All rights reserved.



- Helm 3.2.0+.
- StorageClass disponível com capacidade suficiente para os volumes dos brokers.
- CRDs do Strimzi instalados no cluster (veja seção abaixo).

Gerenciamento de CRDs do Strimzi

Os **Custom Resource Definitions (CRDs)** do Strimzi — como `Kafka`, `KafkaNodePool`, `KafkaTopic`, `KafkaUser` e `KafkaConnect` — são gerenciados separadamente do chart `tdp-kafka`. No TDP Kubernetes, esses CRDs são entregues pelo chart `tdp-crds`.

Os CRDs são recursos de escopo de cluster e **não são removidos** ao desinstalar o chart `tdp-kafka`.

Instalar CRDs antes da primeira instalação do Kafka

Antes de instalar o chart `tdp-kafka` pela primeira vez, instale o chart `tdp-crds` no cluster:

Terminal input

```
helm upgrade --install tdp-crds \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  -n <NAMESPACE> --create-namespace
```

Atualização de versão do Strimzi

Ao atualizar a versão do TDP, atualize o chart `tdp-crds` antes de executar o `helm upgrade` do `tdp-kafka`.

Solução de problemas: conflito de CRD

Se o `helm install` ou `helm upgrade` retornar um erro semelhante a:

```
conflict with "helm" using apiextensions.k8s.io/v1: .spec.versions
```

confirme se o chart `tdp-crds` já foi atualizado para a versão esperada e repita a instalação ou atualização do `tdp-kafka`.

Instalação

© Copyright Tecnisys 2026. All rights reserved.

Instalação (OCI)

Para quem instala ou opera o cluster, o comando abaixo é o primeiro passo declarativo: o Helm aplica o chart `tdp-kafka` no namespace indicado e o Strimzi passa a reconciliar o estado desejado (brokers, Entity Operator, UI e, se habilitado, Connect). Os ajustes de comportamento — capacidade, listeners, replicação, senhas — vêm depois, via `values.yaml` ou `helm upgrade` com o mesmo release.

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka -n <NAMESPACE> --create-
namespace --timeout 10m
```

Validação pós-instalação

Terminal input

```
# Estado do cluster Kafka (recurso Strimzi)
kubectl get kafka -n <NAMESPACE>

# Node Pools do cluster
kubectl get kafkanodepools -n <NAMESPACE>

# Pods gerenciados pelo Strimzi
kubectl get pods -n <NAMESPACE> -l strimzi.io/cluster=<KAFKA_CLUSTER_NAME>

# Serviços gerados (bootstrap, brokers, UI)
kubectl get svc -n <NAMESPACE> | grep kafka

# KafkaConnect instances (Debezium)
kubectl get kafkaconnect -n <NAMESPACE>

# Conectores registrados
kubectl get kafkaconnector -n <NAMESPACE>
```

Parâmetros principais

Parâmetros principais

© Copyright Tecnisys 2026. All rights reserved.



A tabela a seguir resume chaves que costumam ser revisadas na primeira instalação e em upgrades: nome do cluster, versões KRaft, listeners, fatores de replicação, Entity Operator, dimensionamento do Node Pool e opções da Kafka UI. Para valores exatos de todas as chaves do chart, use `helm show values` na versão do pacote que você está implantando; aqui o foco é **o que cada parâmetro controla na prática** para quem opera o release.

Parâmetro	Descrição	Valor padrão
`name`	Nome do cluster Kafka	`tdp-kafka`
`kafkaVersion`	Versão do Kafka	4.1.0
`metadataVersion`	Versão de metadados KRaft	4.1-IV1
`clusterLabel`	Label do cluster	`tdp`
`listeners.plain.enabled`	Listener plain habilitado	`true`
`listeners.plain.port`	Porta do listener plain	`9092`
`listeners.tls.enabled`	Listener TLS habilitado	`true`
`listeners.tls.port`	Porta do listener TLS	`9093`
`config.default.replication.factor`	Fator de replicação padrão dos tópicos	`3`
`config.min.insync.replicas`	Mínimo de réplicas síncronas	`2`
`entityOperator.enabled`	Entity Operator habilitado	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`nodePool.replicas`	Número de brokers/controllers	`4`
`nodePool.storage.volumes[0].size`	Tamanho do volume por nó	`10Gi`
`kafka-ui.enabled`	Kafka UI habilitada	`true`
`kafka-ui.service.type`	Tipo do serviço da UI	`ClusterIP`
`kafka-ui.service.nodePort`	NodePort da UI (quando `type: NodePort`)	`30081`
`config.offsets.topic.replication.factor`	Fator de replicação do tópico de offsets	`3`
`config.transaction.state.log.replication.factor`	Fator de replicação do log de transações	`3`
`nodePool.roles`	Papéis dos nós (controller/broker)	`["controller","broker"]`
`nodePool.storage.type`	Tipo de storage	`jbod`
`security.enabled`	Habilitar autenticação e ACLs no broker	`false`
`security.authentication.type`	Tipo de autenticação Strimzi	`scram-sha-512`
`security.authorization.type`	Tipo de autorização Strimzi	`simple`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`kafka-ui.connection.bootstrapServers`	Bootstrap server da Kafka UI	`tdp-kafka-kafka-bootstrap:9092`
`kafka-ui.applicationConfig.auth.type`	Tipo de autenticação da UI	`LOGIN_FORM`
`kafka-ui.rbac.enabled`	Habilitar RBAC da Kafka UI	`false`
`ingress.annotations`	Anotações do Ingress	`{}`
`TDP-Settings.gateway.ingress.enabled`	Habilita Ingress para a Kafka UI	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita Gateway API (HTTPRoute) para a Kafka UI	`false`
`ingress.ingressClassName`	IngressClass a usar	`nginx`
`ingress.hosts[0].host`	Hostname do Ingress	`kafka-ui.tdp.local`
`gatewayApi.kafkaUi.enabled`	Habilita HTTPRoute para a Kafka UI	`true`
`gatewayApi.kafkaUi.hostnames[0]`	Hostname do HTTPRoute	`kafka-ui.tdp.local`
`nodePool.name`	Nome do Node Pool	`brokers`
`security.listener`	Listener protegido pela autenticação	`plain`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`security.users`	Lista declarativa de `KafkaUser` com ACLs	`[]`
`kafka-ui.service.port`	Porta do serviço da Kafka UI	`80`
`kafka-ui.connection.authentication.existingSecret.name`	Secret com a senha do principal técnico da Kafka UI	`""`
`kafka-ui.rbac.roles`	Roles de RBAC da Kafbat UI	`[]`
`config.transaction.state.log.min.isr`	ISR mínimo para o log de transações	`2`

Detalhes de configuração

Configuração do cluster Kafka

Node Pool

Os Node Pools definem **quantos nós** existem, **que papéis** cada um exerce (metadados KRaft e/ou dados) e **como o armazenamento** se comporta ao escalar ou remover o recurso. Em ambientes pequenos, combinar controller e broker no mesmo pool simplifica a operação; à medida que a carga ou os requisitos de isolamento crescem, separar pools evita que operações de metadados e de I/O de dados disputem os mesmos recursos. As escolhas abaixo impactam diretamente custo de nós, capacidade de disco e o que acontece com os PVCs quando o release é removido.

No Strimzi com KRaft, os nós do cluster são declarados em **Node Pools** (`KafkaNodePool`). Cada pool define a quantidade de réplicas, os papéis exercidos pelos nós e o armazenamento.

© Copyright Tecnisys 2026. All rights reserved.

```
nodePool:
  name: brokers
  replicas: 4
  roles: ["controller", "broker"]
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 10Gi
        deleteClaim: true
        kraftMetadata: shared
```

Campo	Descrição
`roles`	Um nó pode ser `controller`, `broker` ou ambos. Em clusters pequenos, combine os dois papéis. Em clusters grandes, separe-os em pools distintos
`storage.type: jbod`	JBOD (*Just a Bunch of Disks*) permite múltiplos volumes por nó, maximizando throughput de I/O
`deleteClaim: true`	Os PVCs serão excluídos ao remover o NodePool. Defina `false` em produção se quiser preservar os dados
`kraftMetadata: shared`	Indica que este volume também armazena metadados KRaft

DIMENSIONAMENTO EM PRODUÇÃO

Com 4 réplicas combinando controller e broker, o quórum KRaft e a replicação de dados coexistem nos mesmos nós. Para cargas intensas, considere pools separados: um pool de controllers (3 réplicas) e um pool de brokers (quantos forem necessários).

© Copyright Tecnisys 2026. All rights reserved.

Versão e metadados

```
name: tdp-kafka
kafkaVersion: 4.1.0
metadataVersion: 4.1-IV1
clusterLabel: "tdp"
```

O campo `metadataVersion` é a **versão de metadados do KRaft** usada pelo cluster. Mantenha-o alinhado à `kafkaVersion` durante instalações novas. Em atualizações, aumente apenas após todos os brokers estarem na nova versão.

Listeners

Os *listeners* definem os pontos de entrada do cluster para produtores e consumidores.

Manter `plain` e `tls` habilitados é comum: o plain simplifica clientes internos na mesma rede do cluster (menos overhead de TLS), enquanto o TLS atende cenários em que a política de segurança ou a rede exigem criptografia em trânsito. A decisão prática é **qual bootstrap e porta** os aplicativos usarão; misturar sem critério pode gerar configuração inconsistente entre serviços. Os endereços de bootstrap listados mais abaixo são os que você repassa às equipes de aplicação.

```
listeners:
  plain:
    enabled: true
    port: 9092
  tls:
    enabled: true
    port: 9093
```

Listener	Porta	Quando usar
`plain`	9092	Comunicação interna ao cluster, ambientes de desenvolvimento ou redes privadas confiáveis

© Copyright Tecnisys 2026. All rights reserved.

Listener	Porta	Quando usar
`tls`	9093	Comunicação sobre redes não confiáveis ou quando clientes exigem criptografia em trânsito

O nome do serviço de bootstrap segue a convenção Strimzi: `<CLUSTER_NAME>-kafka-bootstrap`. Exemplo com o nome padrão `tdp-kafka`:

- Plain: `tdp-kafka-kafka-bootstrap:9092`
- TLS: `tdp-kafka-kafka-bootstrap:9093`

Replicação e durabilidade

Os valores padrão abaixo priorizam **durabilidade** (mais cópias, `min.insync.replicas` alinhado a `acks=all` em clientes bem configurados) em detrimento de tolerar mais falhas simultâneas sem ajuste. Em troca, clusters com poucos brokers **não** podem honrar fatores de replicação maiores que o número de nós — daí a necessidade de alinhar estes números à topologia real antes de ir a produção.

```
config:
  offsets.topic.replication.factor: 3
  transaction.state.log.replication.factor: 3
  transaction.state.log.min.isr: 2
  default.replication.factor: 3
  min.insync.replicas: 2
```

Estes parâmetros determinam o equilíbrio entre **durabilidade** e **disponibilidade**:

Parâmetro	Significado
`default.replication.factor: 3`	Cada tópico terá 3 cópias distribuídas em brokers distintos
`min.insync.replicas: 2`	Um produtor com `acks=all` só recebe confirmação quando ao menos 2 répli-

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Significado
	cas estiverem sincronizadas. Tolera a perda de 1 broker sem perda de dados
`offsets.topic.replication.factor: 3`	Controla a durabilidade dos offsets dos consumidores (tópico interno `__consumer_offsets`)
`transaction.state.log.*`	Durabilidade do log de transações (produção transacional / exactly-once)

> **CLUSTERS MENORES**
>
> Em ambientes com menos de 3 brokers, reduza `*.replication.factor` e `min.insync.replicas` ao número real de nós. Manter fatores maiores que o número de brokers impede a criação de tópicos.

Entity Operator

```
entityOperator:
  enabled: true
```

O **Entity Operator** é um componente do Strimzi que observa recursos `KafkaTopic` e `KafkaUser` no namespace e os reconcilia com o cluster. Com ele habilitado, tópicos e usuários podem ser gerenciados de forma declarativa como manifests Kubernetes, sem acesso direto à API administrativa do Kafka.

Kafka UI

O Kafka UI oferece uma interface gráfica para inspecionar tópicos, visualizar mensagens, monitorar grupos de consumidores e acompanhar métricas básicas do cluster.

Trate a UI como **ferramenta operacional e de diagnóstico**, não como substituto de controles de acesso do próprio Kafka: ela expõe visão sobre o cluster conforme a configuração de autenticação da aplicação web. O exemplo a seguir usa `PLAINTEXT` em

© Copyright Tecnisys 2026. All rights reserved.

relação ao bootstrap interno — coerente com rede de cluster, mas um alerta a mais para não expor a interface sem senha forte, rede adequada e, quando necessário, os mecanismos descritos na página de segurança.

```
kafka-ui:
  enabled: true
  applicationConfig:
    auth:
      type: LOGIN_FORM
    spring:
      security:
        user:
          name: admin
          password: <PASSWORD>
  connection:
    clusterName: TDP
    bootstrapServers: <CLUSTER_NAME>-kafka-bootstrap:9092
  service:
    type: ClusterIP
    port: 80
    nodePort: 30081
```

Substitua `<CLUSTER_NAME>` pelo valor de `name` definido no cluster (ex.: `tdp-kafka`) e `<PASSWORD>` por uma senha segura.
A conexão da UI com o cluster Kafka é configurada em `kafka-ui.connection.bootstrapServers`.
Para autenticação LDAP ou SCRAM no broker, consulte Segurança — Kafka.

> **CONFIGURAÇÃO INTERNA: `yamlApplicationConfigConfigMap`**
>
> O chart gera automaticamente um ConfigMap (`tdp-kafka-ui-config`) a partir dos valores definidos em `kafka-ui.applicationConfig`. O campo `kafka-ui.yamlApplicationConfig` permanece `null` para que `applicationConfig` tenha precedência. Não é necessário alterar esses campos manualmente.

> **SENHA PADRÃO**

© Copyright Tecnisys 2026. All rights reserved.



> A senha `admin` (padrão do chart) não deve ser mantida em ambientes acessíveis fora da rede interna. Troque antes de expor a interface.

Kafka Connect / Debezium

O chart suporta conectores **Debezium** para captura de dados em tempo real (CDC). A configuração é feita sob a chave `kafkaConnects.*`.

Tipos suportados: `postgres`, `mysql`, `sqlserver`, `oracle`.

Para cada tipo, é possível:

- Gerar o Secret automaticamente com as credenciais: `createSecret: true`
- Referenciar um Secret já existente: `createSecret: false` + `secretName`

Exemplo mínimo para PostgreSQL usando Secret existente:

```
kafkaConnects:
postgres:
  enabled: true
  connect:
    name: connect-debezium-postgres
    replicas: 1
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    kafkaVersion: "4.1.0"
  connectors:
    appdb:
      enabled: true
      name: debezium-postgres-appdb
      topicPrefix: "pgapp"
      postgres:
        createSecret: false
        secretName: "db-pg-appdb"
        host: "tdp-postgresql-project.<NAMESPACE>.svc.cluster.local"
        port: "5432"
        dbname: "appdb"
        user: "debezium"
```

O campo `topicPrefix` define o prefixo dos tópicos Kafka criados pelo conector (por exemplo: `pgapp.public.tabela`).

© Copyright Tecnisys 2026. All rights reserved.

O Strimzi cria um recurso `KafkaConnect` e um `KafkaConnector` para cada conector habilitado.
Verifique o status com:

Terminal input

```
kubectl get kafkaconnect -n <NAMESPACE>
kubectl get kafkaconnector -n <NAMESPACE>
```

> **DOCUMENTAÇÃO DETALHADA**
>
> Consulte Integrações — Kafka para exemplos completos de cada tipo de conector (PostgreSQL, MySQL, SQL Server, Oracle) e gestão de credenciais.

Acesso & Segurança

Acesso

ClusterIP / Port-forward (Kafka UI)

O serviço da Kafka UI usa o tipo `ClusterIP` por padrão.
Para acessar localmente sem expor a porta ao exterior:

Terminal input

```
kubectl port-forward svc/<RELEASE_NAME>-kafka-ui 8080:80 -n <NAMESPACE>
```

Acesse em `http://localhost:8080`.

> **EXEMPLO**
>
> **Exemplo**
>
> ```
> kubectl port-forward svc/tdp-kafka-kafka-ui 8080:80 -n tdp-project
> ```

© Copyright Tecnisys 2026. All rights reserved.

NodePort (Kafka UI)

Para expor via NodePort, defina `kafka-ui.service.type: NodePort` nos values.
A porta padrão é `30081`, configurável em `kafka-ui.service.nodePort`.

Terminal input

```
kubectl get nodes -o wide
```

Acesse em `http://<NODE_IP>:30081`.

Ingress (Kafka UI)

Para expor a Kafka UI via Ingress, habilite `TDP-Settings.gateway.ingress.enabled: true` e configure o bloco `ingress.*` com o IngressClass, hostname e paths desejados. `TDP-Settings.gateway.ingress.enabled` e `TDP-Settings.gateway.gatewayApi.enabled` são **mutuamente exclusivos** — ative somente um.

> **KAFKA UI ≠ BROKERS KAFKA**
>
> O Ingress publica apenas a interface web da Kafka UI.
> Produtores e consumidores externos continuam precisando de configuração adicional para acessar os brokers (listeners Kafka).

Consulte Exposição externa — Kafka UI para o exemplo completo com YAML, IngressClass (incluindo OpenShift), `ingress.tls` e validação.

Gateway API (Kafka UI)

Para expor a Kafka UI via Kubernetes Gateway API, habilite `TDP-Settings.gateway.gatewayApi.enabled: true` e configure o bloco `gatewayApi.*`.
O chart cria automaticamente os recursos Gateway e HTTPRoute quando habilitado.

> **KAFKA UI ≠ BROKERS KAFKA**
>
> O Gateway API publica apenas a interface web da Kafka UI.
> Produtores e consumidores externos continuam precisando de configuração adicional para acessar os brokers (listeners Kafka).

© Copyright Tecnisys 2026. All rights reserved.

Consulte Exposição externa — Kafka UI para o exemplo completo com YAML, `gatewayClassName`, listeners, HTTPRoute, `/etc/hosts` e validação.

Bootstrap para produtores e consumidores

O endereço de bootstrap segue a convenção Strimzi: `<KAFKA_CLUSTER_NAME>-kafka-bootstrap.<NAMESPACE>.svc.cluster.local`.

Com o valor padrão `name: tdp-kafka`:

```
# Plain (PLAINTEXT) — porta padrão 9092
tdp-kafka-kafka-bootstrap:9092

# TLS — porta padrão 9093
tdp-kafka-kafka-bootstrap:9093
```

Autenticação da UI

> **DOCUMENTAÇÃO DETALHADA**
>
> Consulte Segurança — Kafka para configurar autenticação `LOGIN_FORM` ou `LDAP` na interface web.

Solução de problemas

Solução de problemas

Terminal input

```
# Detalhes e eventos do cluster Kafka
kubectl describe kafka <KAFKA_CLUSTER_NAME> -n <NAMESPACE>

# Logs de um broker
kubectl logs -n <NAMESPACE> \
  -l strimzi.io/cluster=<KAFKA_CLUSTER_NAME> \
  -c kafka

# Logs da Kafka UI
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/name=kafka-ui
```

© Copyright Tecnisys 2026. All rights reserved.

```
# Detalhes de um conector com problema
kubectl describe kafkaconnector <CONNECTOR_NAME> -n <NAMESPACE>
```

Problemas comuns

Problema	Causa provável	Ação
Conflito de CRD na instalação (`spec.versions`)	CRDs Strimzi já existem no cluster	Confirme que o chart `tdp-crds` está na versão esperada e repita o `helm upgrade` do `tdp-kafka`
Pods em `Pending`	StorageClass inexistente ou sem capacidade	`kubectl get storageclass` e `kubectl describe pvc -n <NAMESPACE>`
Kafka UI não conecta ao cluster	`bootstrapServers` incorreto ou credenciais Kafka	Confira `kafka-ui.connection.bootstrapServers` e o Secret referenciado em `kafka-ui.connection.authentication.existingSecret`
Broker não inicia	`min.insync.replicas` > número de brokers disponíveis	Ajuste `config.min.insync.replicas` ao número real de réplicas
Conector em estado `FAILED`	Credenciais ou host do banco incorretos	`kubectl describe kafkaconnector <CONNECTOR_NAME> -n <NAMESPACE>`
Kafka UI não inicia (RBAC)	Recursos ou ações RBAC incompatíveis com a versão da UI	Verifique `kafka-ui.rbac.roles` e os logs do pod
Pods com `OOMKilled`	Limites de CPU/memória insuficientes	Ajuste os limites de recursos dos pods via `resource`

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Ação
		`s` do subchart

Comandos de diagnóstico

Terminal input

```
# Baixe o chart antes de validar:
helm pull oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka --untar
helm lint tdp-kafka/
helm template tdp-kafka tdp-kafka/ -n <NAMESPACE>

# Estado do cluster e usuários
kubectl -n <NAMESPACE> get kafka
kubectl -n <NAMESPACE> get kafkauser

# Secrets gerados pelo Strimzi
kubectl -n <NAMESPACE> get secret

# Logs do operador Strimzi
kubectl -n <NAMESPACE> logs deployment/strimzi-cluster-operator

# Logs da Kafka UI
kubectl -n <NAMESPACE> logs deployment/<RELEASE_NAME>-kafka-ui

# Conectores
kubectl -n <NAMESPACE> get kafkaconnectors
kubectl -n <NAMESPACE> describe kafka <KAFKA_CLUSTER_NAME>
```

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

PVCS

Os PVCs dos brokers **não são removidos** automaticamente pelo `helm uninstall`, salvo se `nodePool.storage.volumes[].deleteClaim: true` estiver configurado (padrão do chart).
Para remover PVCs manualmente (operação **destrutiva** e irreversível):

Terminal input

```
kubectl delete pvc -n <NAMESPACE> -l strimzi.io/cluster=
<KAFKA_CLUSTER_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.

Kafka – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 4.1.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Este documento cobre autenticação e autorização do **cluster Kafka** (brokers) e da **Kafka UI**.
Para listeners plain/TLS, consulte Configuração do Kafka.

Modelo de segurança

A autenticação da Kafka UI e a autenticação no broker Kafka são **preocupações separadas**:

- um usuário humano autentica-se na Kafka UI via `kafka-ui.applicationConfig.auth` (por exemplo `LOGIN_FORM` ou `LDAP`);
- o backend da Kafka UI conecta-se ao Kafka usando o principal técnico configurado em `kafka-ui.connection.authentication`;
- os brokers Kafka avaliam ACLs para o **principal Kafka** usado pela aplicação, não para o usuário humano que fez login na UI.

Exemplo:

- o usuário LDAP `fry` faz login na Kafka UI;
- a Kafka UI conecta-se ao Kafka como `kafka-ui`;
- criação de tópicos, leitura de mensagens e ações administrativas são avaliadas pelo Kafka como principal `kafka-ui`.

Isso significa que as decisões de **auditoria e autorização** do Kafka baseiam-se no usuário Kafka configurado para a UI (`kafka-ui`), não no nome de usuário LDAP que fez login.

Autenticação no broker (SCRAM-SHA-512)

© Copyright Tecnisys 2026. All rights reserved.

Segurança desabilitada (padrão)

Por padrão, autenticação e autorização estão desabilitadas:

```
security:
  enabled: false
  authentication:
    enabled: false
  authorization:
    enabled: false
```

Os listeners plain e TLS operam sem autenticação. Adequado apenas para ambientes internos controlados.

Habilitar SCRAM-SHA-512 e ACLs simples

Para habilitar autenticação SCRAM-SHA-512 com ACLs simples:

```
security:
  enabled: true
  listener: plain
  authentication:
    enabled: true
    type: scram-sha-512
  authorization:
    enabled: true
    type: simple
    superUsers:
      - platform-admin
```

Com `security.enabled: true`, o Strimzi cria recursos `KafkaUser` declarativos e aplica ACLs conforme `security.users`.

Usuário produtor (exemplo)

```
security:
  users:
    - name: producer-app
      acls:
        - resource:
            type: topic
```

© Copyright Tecnisys 2026. All rights reserved.

```
            name: orders.events
            patternType: literal
          operations:
            - Write
            - Create
            - Describe
        - resource:
            type: cluster
          operations:
            - IdempotentWrite
```

Usuário consumidor (exemplo)

```
security:
  users:
    - name: consumer-app
      acls:
        - resource:
            type: topic
            name: orders.events
            patternType: literal
          operations:
            - Read
            - Describe
        - resource:
            type: group
            name: orders-consumer
            patternType: literal
          operations:
            - Read
```

Superusuário (exemplo)

```
security:
  authorization:
    superUsers:
      - admin-user
  users:
    - name: admin-user
```

© Copyright Tecnisys 2026. All rights reserved.

A entrada `admin-user` é renderizada como `KafkaUser`; o acesso de superusuário é concedido por `security.authorization.superUsers`. Superusuários não precisam de ACLs.

O Strimzi gera um Secret com credenciais para cada `KafkaUser`:

Terminal input

```
kubectl -n <NAMESPACE> get kafkauser producer-app -o yaml
kubectl -n <NAMESPACE> get secret producer-app -o jsonpath='{.data.password}' |
base64 -d
```

Propriedades típicas de cliente SCRAM:

```
bootstrap.servers=tdp-kafka-kafka-bootstrap:9092
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required username="producer-app" password="<PASSWORD>";
```

Kafka UI — autenticação web

A Kafka UI suporta dois modos de autenticação web:

Modo	Descrição
`LOGIN_FORM`	Usuário e senha definidos em `kafka-ui.applicationConfig`
`LDAP`	Autenticação delegada a um servidor LDAP/AD corporativo

LOGIN_FORM (padrão)

```
kafka-ui:
  applicationConfig:
    auth:
```

© Copyright Tecnisys 2026. All rights reserved.

```
      type: LOGIN_FORM
    spring:
      security:
        user:
          name: admin
          password: <PASSWORD>
```

LDAP (opcional)

O chart oferece **duas formas** de fornecer a senha de bind LDAP da Kafka UI. Em ambas, a autenticação web é delegada ao servidor LDAP — a diferença está em **como a senha de bind chega ao pod**.

Forma 1 — senha inline no values (não recomendada)

> **EVITE EM PRODUÇÃO**
>
> Esta forma grava a senha de bind em texto plano no arquivo de valores. Use apenas em testes isolados — nunca em ambientes acessíveis nem em arquivos de values versionados no Git.

```
kafka-ui:
  applicationConfig:
    auth:
      type: LDAP
    spring:
      ldap:
        urls: ldap://<LDAP_HOST>:389
        base: "<LDAP_USER_BASE_DN>"
        admin-user: "<LDAP_BIND_DN>"
        admin-password: "<LDAP_BIND_PASSWORD>"
    management:
      health:
        ldap:
          enabled: true
```

Forma 2 — senha via Secret (recomendada)

A senha de bind LDAP não deve ficar em texto plano no arquivo de valores. Crie um Secret e injete-a via `kafka-ui.envs.secretMappings`. A Kafka UI é uma aplicação Spring

© Copyright Tecnisys 2026. All rights reserved.

Boot — a variável de ambiente `SPRING_LDAP_ADMIN_PASSWORD` sobrescreve a propriedade `spring.ldap.admin-password` em tempo de execução.

Crie o Secret antes de instalar o chart:

Terminal input

```
kubectl create secret generic kafka-ui-ldap-secret \
  --from-literal=admin-password=<LDAP_BIND_PASSWORD> \
  -n <NAMESPACE>
```

```
kafka-ui:
  enabled: true
  yamlApplicationConfig: null
  yamlApplicationConfigConfigMap:
    name: tdp-kafka-ui-config
    keyName: config.yml
  applicationConfig:
    auth:
      type: LDAP
    spring:
      ldap:
        urls: ldap://<LDAP_HOST>:389
        base: "<LDAP_USER_BASE_DN>"
        admin-user: "<LDAP_BIND_DN>"
    management:
      health:
        ldap:
          enabled: true
  envs:
    secretMappings:
      SPRING_LDAP_ADMIN_PASSWORD:
        name: kafka-ui-ldap-secret
        keyName: admin-password
```

Campo	Descrição	Exemplo
`urls`	Endereço do servidor LDAP	`ldap://ldap.empresa.com.br:389`

© Copyright Tecnisys 2026. All rights reserved.

Campo	Descrição	Exemplo
`base`	Base DN de busca de usuários	`dc=empresa,dc=com,dc=br`
`admin-user`	DN do usuário de bind	`cn=kafkaui,dc=empresa,dc=com,dc=br`
`admin-password`	Senha do bind — **injetada via `envs.secretMappings`** (Secret `kafka-ui-ldap-secret`), nunca inline no values	—
`management.health.ldap.enabled`	Habilita o health-check LDAP da Kafka UI	`true`

Para LDAPS (TLS), substitua `ldap://` por `ldaps://` e use a porta 636.

> **CONFIGURAÇÃO COMPLETA**
>
> Para um deploy com broker seguro (`security.enabled: true`), combine este bloco de autenticação da UI com o bloco da seção **Kafka UI — conexão com o broker** (abaixo), no mesmo arquivo de values.

Kafka UI — conexão com o broker

Quando `security.enabled=true`, crie um `KafkaUser` correspondente ao principal técnico da UI.
O Strimzi gera um Secret com o mesmo nome do usuário por padrão.

> **SASL AUTOMÁTICO**
>
> Quando `security.enabled=true`, o chart injeta automaticamente `security.protocol`, `sasl.mechanism` e `sasl.jaas.config` na conexão da Kafka

© Copyright Tecnisys 2026. All rights reserved.

UI com o broker (`kafka-ui.connection`). Não é necessário definir essas propriedades manualmente.

```
security:
  enabled: true
  authentication:
    enabled: true
  authorization:
    enabled: true

kafka-ui:
  connection:
    clusterName: TDP
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    authentication:
      username: kafka-ui
      existingSecret:
        name: kafka-ui
        passwordKey: password
```

Verifique o Secret gerado:

Terminal input

```
kubectl -n <NAMESPACE> get kafkauser kafka-ui -o yaml
kubectl -n <NAMESPACE> get secret kafka-ui -o yaml
```

RISCO DE SUPERUSUÁRIO

Se `kafka-ui` for promovido a `superUser`, todo acesso administrativo via UI será executado com privilégios totais no broker.
Restrinja o acesso à Kafka UI a um conjunto pequeno de usuários confiáveis nesse cenário.

Propriedades típicas de cliente SCRAM para consumidores:

© Copyright Tecnisys 2026. All rights reserved.



```
bootstrap.servers=tdp-kafka-kafka-bootstrap:9092
group.id=orders-consumer
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required username="consumer-app" password="<PASSWORD>";
```

RBAC da Kafka UI

O bloco `kafka-ui.rbac` injeta a seção `rbac` no `config.yml` da Kafbat UI, permitindo restringir ações da UI por usuário LDAP — sem conceder acesso administrativo a todos os usuários autenticados.

Modelo recomendado:

- apenas um pequeno conjunto de usuários LDAP confiáveis deve ter permissões completas na UI;
- todos os demais usuários da Kafka UI devem receber permissões somente-leitura;
- se `kafka-ui` estiver configurado como `superUser` do Kafka, o RBAC da UI deve estar **sempre habilitado**.

```
kafka-ui:
  rbac:
    enabled: true
    roles:
      - name: admins
        clusters:
          - TDP
        subjects:
          - provider: ldap
            type: user
            value: fry
        permissions:
          - resource: topic
            value: ".*"
            actions: all
      - name: readonly
        clusters:
          - TDP
        subjects:
          - provider: ldap
```

© Copyright Tecnisys 2026. All rights reserved.

```
            type: user
            value: leela
        permissions:
          - resource: topic
            value: ".*"
            actions:
              - view
              - messages_read
```

Com esse modelo, `fry` usa a UI completa e `leela` pode apenas visualizar tópicos e ler mensagens — mas o Kafka ainda vê ambas as operações como principal `kafka-ui`.

Produtores e consumidores externos

Aplicações externas devem autenticar-se diretamente no Kafka com identidades próprias, **sem reutilizar** o principal técnico `kafka-ui`.

Recomendações:

- crie um `KafkaUser` por aplicação ou fronteira de segurança;
- conceda apenas as ACLs necessárias;
- deixe o Strimzi gerar e gerenciar credenciais sempre que possível;
- distribua o Secret gerado de forma segura ao runtime da aplicação.

> **LISTENERS INTERNOS POR PADRÃO**
>
> Os exemplos abaixo assumem que o cliente alcança o listener Kafka selecionado. Este chart expõe **listeners internos por padrão** — o acesso a partir de fora do cluster exige exposição adicional de listener, não coberta por estes exemplos.

Autenticação TLS (cliente)

Para autenticação TLS de cliente, defina o usuário com `authentication.type: tls` e conecte-se via listener TLS:

```
security:
  listener: tls
  users:
    - name: consumer-tls
```

© Copyright Tecnisys 2026. All rights reserved.

```
    authentication:
      type: tls
    acls:
      - resource:
          type: topic
          name: orders.events
          patternType: literal
        operations:
          - Read
          - Describe
      - resource:
          type: group
          name: orders-consumer
          patternType: literal
        operations:
          - Read
```

O Secret gerado contém `ca.crt`, `user.crt`, `user.key`, `user.p12` e `user.password`.

Propriedades típicas de cliente TLS usando arquivos PEM:

```
bootstrap.servers=tdp-kafka-kafka-bootstrap:9093
security.protocol=SSL
ssl.truststore.type=PEM
ssl.keystore.type=PEM
ssl.truststore.certificates=/etc/kafka-user/ca.crt
ssl.keystore.certificate.chain=/etc/kafka-user/user.crt
ssl.keystore.key=/etc/kafka-user/user.key
```

Orientações de ACL para produtores e consumidores

Use um principal Kafka por aplicação ou tipo de carga. Evite compartilhar um principal amplo entre vários sistemas.

Tipo	ACLs típicas
Produtor	`Write`, `Create`, `Describe` no(s) tópico(s); `IdempotentWrite` no `cluster` para produção idempotente

© Copyright Tecnisys 2026. All rights reserved.

Tipo	ACLs típicas
Consumidor	`Read`, `Describe` nos tópicos; `Read` no consumer group
Ferramenta admin / manutenção	Identidade separada; evite reutilizar credenciais de produtor/consumidor
Superusuário	Somente quando estritamente necessário; não usar `All` desnecessariamente

Não versionar senhas SCRAM em `values.yaml` público. Prefira `createSecret: false` com Secrets externos em produção.

Recomendações

Aspecto	Recomendação
Ambiente de desenvolvimento	`LOGIN_FORM` com senha forte, acesso via port-forward
Ambiente compartilhado / produção	`LDAP` com LDAPS (porta 636)
Broker em produção	`security.enabled: true` com SCRAM-SHA-512 e ACLs mínimas
Credenciais	Nunca versionar senhas em `values.yaml` público
Exposição da UI	Prefira Ingress com TLS ou port-forward controlado

Solução de problemas

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
Credenciais inválidas na UI (LOGIN_FORM)	Senha incorreta ou typo em `name`/`password`	Verificar `kafka-ui.applicationConfig.spring.security.user`
Login LDAP falha com "invalid credentials"	Bind DN ou senha incorretos	Testar bind com `ldapsearch` externo
Login LDAP falha com "connection refused"	Host/porta LDAP inacessível pelo pod	Verificar conectividade de rede
Kafka UI não conecta ao broker	Secret ou bootstrap incorretos	Verificar `kafka-ui.connection` e Secret do `KafkaUser`
Falha de startup por RBAC	Recursos ou ações RBAC incompatíveis com a versão da UI	Verificar `kafka-ui.rbac.roles` e logs do pod
Mudanças no `applicationConfig` não refletem	Pod não reiniciado após atualização	`kubectl -n <NAMESPACE> rollout restart deployment <RELEASE_NAME>-kafka-ui`

© Copyright Tecnisys 2026. All rights reserved.

Kafka – Integração

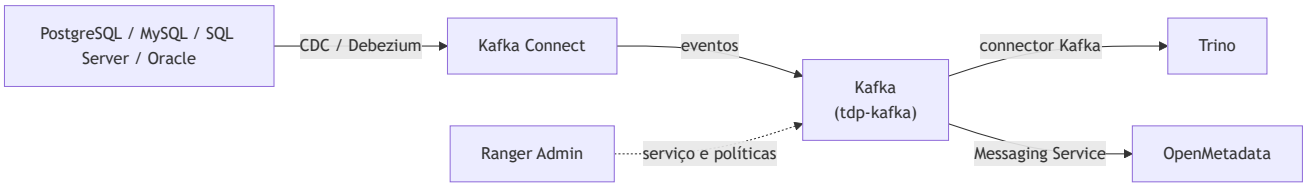
CHART

Version 3.0.1 | Type application | AppVersion 4.1.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



Esta página descreve como configurar integrações com o Apache Kafka instalado via chart `tdp-kafka`.
O chart implanta um cluster Kafka gerenciado pelo operador Strimzi e, opcionalmente, conectores Debezium para captura de dados em tempo real (**Change Data Capture — CDC**).

Para configurações de autenticação e segurança (LDAP, TLS, LDAP na Kafka UI), consulte Segurança — Kafka.

Além do CDC via Debezium, o Kafka também se integra a outros três componentes da plataforma:

- **Trino**: consulte o conector Kafka para consultar tópicos via SQL.
- **OpenMetadata**: o Kafka pode ser registrado como *Messaging Service* — consulte Integrações — OpenMetadata.
- **Ranger**: políticas de acesso aos tópicos podem ser centralizadas via `rangerIntegrations.kafka` — consulte Integrações — Ranger.

Kafka Connect e Debezium

O chart `tdp-kafka` suporta a criação de clusters **Kafka Connect** com conectores **Debezium** para CDC de bancos de dados relacionais.

© Copyright Tecnisys 2026. All rights reserved.



O Debezium captura eventos de alteração (INSERT, UPDATE, DELETE) do banco de dados e os publica em tópicos Kafka.

Bancos de dados suportados

Tipo	Chave em `kafkaConnects`	Banco
PostgreSQL	`kafkaConnects.postgres`	PostgreSQL
MySQL	`kafkaConnects.mysql`	MySQL / MariaDB
SQL Server	`kafkaConnects.sqlserver`	Microsoft SQL Server
Oracle	`kafkaConnects.oracle`	Oracle Database

Como habilitar um conector

Para cada banco de dados que deseja monitorar:

1. Habilite o tipo de conector com `kafkaConnects.<CONNECTOR_TYPE>.enabled: true`.
2. Configure o cluster Kafka Connect em `kafkaConnects.<CONNECTOR_TYPE>.connect`.
3. Configure os conectores individuais em `kafkaConnects.<CONNECTOR_TYPE>.connectors`.
4. Defina as credenciais do banco usando `createSecret` ou `secretName`.

Endereço interno do cluster Kafka

O parâmetro `bootstrapServers` define o endereço interno do cluster Kafka ao qual o Kafka Connect se conecta.
Quando o chart `tdp-kafka` é instalado com o release name `tdp-kafka` no mesmo namespace, o endereço padrão é:

```
bootstrapServers: tdp-kafka-kafka-bootstrap:9092
```

Se o release name ou namespace forem diferentes, ajuste o valor de `bootstrapServers` de acordo.

Gerenciamento de credenciais

© Copyright Tecnisys 2026. All rights reserved.

O chart oferece duas formas de fornecer credenciais do banco de dados ao conector:

Opção	Parâmetros	Recomendação
Secret criado pelo chart	`createSecret: true` + `user` + `password`	Somente em ambientes de teste; evitar em produção
Secret existente	`createSecret: false` + `secretName`	Recomendado para produção

> **NÃO VERSIONE SENHAS EM `values.yaml`**
>
> Ao usar `createSecret: true`, a senha é fornecida como parâmetro Helm. Nunca commite senhas em texto plano em arquivos `values.yaml` versionados no Git.
> Em produção, prefira `createSecret: false` com um Kubernetes Secret preexistente referenciado via `secretName`.

Exemplo: conector PostgreSQL com Secret existente

Este exemplo usa um Kubernetes Secret preexistente para as credenciais do banco, evitando senhas em texto plano no `values.yaml`.

1. Crie o Secret com as credenciais do banco:

Terminal input

```
kubectl -n <NAMESPACE> create secret generic <DEBEZIUM_SECRET_NAME> \
  --from-literal=username='<DATABASE_USER>' \
  --from-literal=password='<PASSWORD>'
```

2. Configure o conector no `values.yaml`:

```
kafkaConnects:
postgres:
```

© Copyright Tecnisys 2026. All rights reserved.

```
  enabled: true
  connect:
    name: connect-debezium-postgres
    replicas: 1
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    kafkaVersion: "4.1.0"
  connectors:
    appdb:
      enabled: true
      name: debezium-postgres-<APPLICATION_NAME>
      topicPrefix: "<TOPIC_PREFIX>"
      postgres:
        createSecret: false
        secretName: "<DEBEZIUM_SECRET_NAME>"
        host: "<DATABASE_HOST>.<DB_NAMESPACE>.svc.cluster.local"
        port: "5432"
        dbname: "<DATABASE_NAME>"
```

> **NOTE**
>
> - Substitua `<TOPIC_PREFIX>` pelo prefixo desejado para os tópicos Kafka gerados pelo conector (ex.: `pgapp`).
> - O campo `host` deve referenciar o serviço Kubernetes do banco de dados — normalmente no formato `<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local`.
> - `kafkaVersion` deve ser compatível com a versão do cluster Kafka instalado (ver `appVersion` no `Chart.yaml` do `tdp-kafka`).

Exemplo: conector MySQL com Secret existente

```
kafkaConnects:
mysql:
  enabled: true
  connect:
    name: connect-debezium-mysql
    replicas: 1
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    kafkaVersion: "4.1.0"
  connectors:
    appdb:
      enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
      name: debezium-mysql-<APPLICATION_NAME>
      topicPrefix: "<TOPIC_PREFIX>"
      mysql:
        createSecret: false
        secretName: "<DEBEZIUM_SECRET_NAME>"
        host: "<DATABASE_HOST>.<DB_NAMESPACE>.svc.cluster.local"
        port: "3306"
        databaseIncludeList: "<DATABASE_NAME>"
```

Exemplo: conector SQL Server com Secret existente

```
kafkaConnects:
sqlserver:
  enabled: true
  connect:
    name: connect-debezium-sqlserver
    replicas: 1
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    kafkaVersion: "4.1.0"
  connectors:
    appdb:
      enabled: true
      name: debezium-sqlserver-<APPLICATION_NAME>
      topicPrefix: "<TOPIC_PREFIX>"
      sqlserver:
        createSecret: false
        secretName: "<DEBEZIUM_SECRET_NAME>"
        host: "<DATABASE_HOST>.<DB_NAMESPACE>.svc.cluster.local"
        port: "1433"
        databaseNames: "<DATABASE_NAME>"
```

Exemplo: conector Oracle com Secret existente

```
kafkaConnects:
oracle:
  enabled: true
  connect:
    name: connect-debezium-oracle
    replicas: 1
    bootstrapServers: tdp-kafka-kafka-bootstrap:9092
    kafkaVersion: "4.1.0"
  connectors:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    appdb:
      enabled: true
      name: debezium-oracle-<APPLICATION_NAME>
      topicPrefix: "<TOPIC_PREFIX>"
      oracle:
        createSecret: false
        secretName: "<DEBEZIUM_SECRET_NAME>"
        host: "<DATABASE_HOST>.<DB_NAMESPACE>.svc.cluster.local"
        port: "1521"
        dbname: "<CDB_NAME>"
        pdbName: "<PDB_NAME>"
```

Validação dos conectores

Após a instalação, verifique se os recursos foram criados corretamente:

Terminal input

```
kubectl get kafkaconnect -n <NAMESPACE>
kubectl get kafkaconnector -n <NAMESPACE>
```

Para ver os detalhes de um conector específico:

Terminal input

```
kubectl describe kafkaconnector <CONNECTOR_NAME> -n <NAMESPACE>
```

Para verificar os logs do pod Kafka Connect:

Terminal input

```
kubectl logs -n <NAMESPACE> -l strimzi.io/cluster=<CONNECT_NAME> --tail=100
```

> **PRÉ-REQUISITO: CRDS STRIMZI**
>
> Os recursos `KafkaConnect` e `KafkaConnector` são CRDs do Strimzi. Certifique-se de que o chart `tdp-crds` está instalado antes de habilitar

© Copyright Tecnisys 2026. All rights reserved.

conectores Debezium.
Consulte Instalação via Helm para a ordem de instalação dos charts.

© Copyright Tecnisys 2026. All rights reserved.

NiFi

CHART

Version 3.0.1 | Type application | AppVersion 1.28.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | Ingress suportado | Gateway API suportado

O chart `tdp-nifi` implanta o Apache **NiFi** com o operador **NiFiKop** e **ZooKeeper**.

Na operação diária, quem mantém o NiFi lida com um *cluster* declarado no CRD `NifiCluster`: alterações em ZooKeeper, certificados TLS, *listeners* ou Ingress repercutem diretamente na coordenação entre nós e na URL que os usuários utilizam para acessar a interface. O operador reconcilia o estado desejado — se o CRD estiver inconsistente ou um pré-requisito (por exemplo cert-manager) falhar, o sintoma costuma ser `NifiCluster` preso em estado intermediário em vez de um único *Pod* com erro genérico.

Visão geral

Propriedade	Valor
Chart	`tdp-nifi`
Versão do NiFi	1.28.0
Versão do chart	3.0.1
Operador	NiFiKop
Coordenação	ZooKeeper embutido como subchart

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Recurso principal	`NifiCluster`
Recursos suportados	LDAP, S3, Ingress e Gateway API
Acesso padrão	Service `NodePort` para a UI do NiFi

Páginas relacionadas

- Segurança — NiFi — LDAP, TLS, modos de segurança e autorização Ranger
- Integrações — NiFi — registro da integração com Apache Ranger
- Exposição externa: Ingress e Gateway API — exposição HTTP/HTTPS da UI

O que é o NiFiKop?

O NiFiKop é um operador Kubernetes especializado no ciclo de vida de clusters Apache NiFi.

Em vez de gerenciar instâncias NiFi manualmente, você descreve o estado desejado em um recurso Kubernetes chamado **`NifiCluster`**, e o NiFiKop cuida de criar, monitorar e atualizar os nós NiFi de forma declarativa.

O chart `tdp-nifi` implanta tanto o operador NiFiKop quanto o cluster NiFi em si, incluindo:

Componente	Função
NiFiKop Operator	Gerencia o ciclo de vida do NifiCluster (CRD)
ZooKeeper	Coordenação de cluster entre os nós NiFi
NifiCluster	Recurso que define o cluster: nós, listeners, serviços externos

© Copyright Tecnisys 2026. All rights reserved.

Componente	Função
cert-manager	Emite certificados TLS para comunicação segura entre nós (quando `webhook.enabled=true`)

Esse modelo baseado em operador significa que a maioria das configurações do NiFi vai para `nifiCluster.*` em vez de parâmetros Helm diretos — o operador traduz esses valores em ConfigMaps, Secrets e Pods com a configuração correta do NiFi.

NiFi Cluster

O NiFi é implantado por meio de um recurso customizado `NifiCluster`. A configuração principal do cluster fica sob `nifiCluster`, incluindo nós, grupos de configuração, serviços externos, listeners, storage e segurança.

O chart padrão inicia com um único nó (`nifiCluster.nodes` contém apenas `id: 1`). Para executar nós adicionais, adicione entradas como `id: 2` e `id: 3`, apontando para o `nodeConfigGroup` desejado no arquivo de valores ou via `--set nifiCluster.nodes[1].id=2`:

```
nifiCluster:
  nodes:
    - id: 1
      nodeConfigGroup: "default_group"
    - id: 2
      nodeConfigGroup: "default_group"
    - id: 3
      nodeConfigGroup: "default_group"
```

ZooKeeper

O ZooKeeper é implantado como subchart e usado pelo NiFi para coordenação do cluster. Em instalações pequenas, o valor padrão `zookeeper.replicaCount: 1` atende cenários simples; para ambientes de maior criticidade, ajuste réplicas, persistência e recursos conforme a política do cluster.

> **PARA SABER MAIS**

© Copyright Tecnisys 2026. All rights reserved.



> Consulte Apache NiFi — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

O chart `tdp-nifi` distribui a configuração nos blocos abaixo:

- **`tdp-nifi:`** — habilitação do chart.
- **`global:`** — configurações globais compartilhadas por dependências.
- **`cert-manager:`** — subchart opcional usado quando o webhook do NiFiKop exige certificados.
- **`zookeeper:`** — ZooKeeper usado para coordenação do cluster NiFi.
- **`nifikop:`** — operador NiFiKop, incluindo imagem, webhook, cert-manager e recursos.
- **`nifiCluster:`** — definição principal do cluster NiFi: segurança, configuração somente leitura, serviços, nós, listeners, storage e autorização.
- **`TDP-Settings:`**, **`ingress:`** e **`gatewayApi:`** — controle de exposição da UI por Ingress ou Gateway API.

```
tdp-nifi:
  enabled: true

cert-manager:
  enabled: false

zookeeper:
  replicaCount: 1

nifikop:
  enabled: true
  webhook:
    enabled: false

nifiCluster:
  nodes:
    - id: 1
      nodeConfigGroup: "default_group"

TDP-Settings:
  gateway:
    ingress:
```

© Copyright Tecnisys 2026. All rights reserved.

```
      enabled: false
    gatewayApi:
      enabled: false
```

Imagens do NiFi

As imagens do NiFi são unificadas no registry `registry.tecnisys.com.br/tdp/images/nifi`, com uma variante de tag por plataforma (base 1.28.0):

Plataforma	Tag da imagem
Kubernetes	1.28.0-0
OpenShift	1.28.0-openshift

Defina a imagem do cluster em `nifiCluster.clusterImage`:

```
nifiCluster:
clusterImage: "registry.tecnisys.com.br/tdp/images/nifi:1.28.0-0"
```

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+
- Helm 3.2.0+

Recursos opcionais

- **Ingress Controller** — necessário apenas quando `ingress.enabled=true`.
- **cert-manager** — necessário apenas quando `nifikop.webhook.enabled=true`.

> **NOTA**
>
> - Com `ingress.enabled=true`, o host configurado em `ingress.rules` só recebe tráfego quando há um Ingress Controller disponível no cluster.

© Copyright Tecnisys 2026. All rights reserved.



- Com `nifikop.webhook.enabled=true`, os controllers do cert-manager precisam estar instalados e em execução para que o webhook do NiFiKop consiga emitir certificados.

CRDs do NiFiKop e cert-manager (tdp-crds)

Antes da primeira instalação do `tdp-nifi`, instale o chart `tdp-crds`, que entrega os CRDs usados pelo NiFiKop e, quando aplicável, os CRDs do cert-manager:

Terminal input

```
helm upgrade --install tdp-crds \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  --namespace tdp --create-namespace
```

Quando `nifikop.webhook.enabled=true`, além dos CRDs, os controllers do cert-manager também precisam estar instalados e em execução (`cert-manager`, `cainjector` e `webhook`). Se os controllers ainda não estiverem instalados, instale o cert-manager antes de habilitar o webhook do NiFiKop. Consulte Segurança — NiFi para detalhes.

OpenShift: runAsUser e fsGroup

No OpenShift, o SCC `restricted` exige que os pods executem com UID/GID atribuído ao namespace. Antes do deploy, obtenha o intervalo de UID do namespace e configure:

Terminal input

```
uid=$(kubectl get namespace <NAMESPACE> -
o=jsonpath='{.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}'
| sed 's/\/10000$//' | tr -d '[:space:]')
```

Aplique no deploy:

Terminal input

```
helm upgrade --install tdp-nifi oci://registry.tecnisys.com.br/tdp/charts/tdp-
nifi \
  -n <NAMESPACE> --create-namespace \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  --set nifiCluster.nodeConfigGroups.default_group.runAsUser="${uid}" \
  --set nifiCluster.nodeConfigGroups.default_group.fsGroup="${uid}"
```

Ou no `values.yaml`:

```
nifiCluster:
  nodeConfigGroups:
    default_group:
      runAsUser: "<UID>"
      fsGroup: "<UID>"
```

Instalação

Instalação (OCI)

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-nifi \
  -n <NAMESPACE> --create-namespace
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão (referência)
`tdp-nifi.enabled`	Habilitar o chart	`true`
`zookeeper.replicaCount`	Réplicas ZooKeeper	`1`
`zookeeper.persistence.storageClass`	StorageClass do ZooKeeper	`local-path`
`zookeeper.persistence.size`	Tamanho do volume do ZooKeeper	`4Gi`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão (referência)
`nifikop.enabled`	Operador NiFiKop	`true`
`nifikop.watchOwnNamespace`	Watch só no namespace do release	`true`
`nifikop.webhook.enabled`	Webhook do operador	`false`
`nifikop.certManager.enabled`	cert-manager para webhook	`false`
`nifiCluster.zkAddress`	Endereço ZooKeeper	`<RELEASE_NAME>-zookeeper:2181`
`nifiCluster.externalServices[0].spec.type`	Tipo do serviço NiFi	`NodePort`
`nifiCluster.externalServices[0].spec.portConfigs[0].port`	Porta HTTP dos nós NiFi	`8080`
`nifiCluster.nodeConfigGroups.default_group.resourcesRequirements`	Recursos (CPU/memória) dos nós NiFi	execute `helm show values`
`ingress.enabled`	Criar Ingress	`false`
`ingress.rules[0].host`	Host do Ingress	`nifi.tdp.local`
`ingress.ingressClassName`	Classe do Ingress	`nginx`
`TDP-Settings.gateway.ingress.enabled`	Habilita Ingress	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão (referência)
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita Gateway API (HTTPRoute)	`false`
`gatewayApi.nifi.enabled`	Habilita HTTPRoute para o NiFi	`false`
`gatewayApi.nifi.hostnames[0]`	Hostname do HTTPRoute	`nifi.tdp.local`

Ajuste imagens, recursos, `NifiCluster`, listeners e health checks no arquivo de valores; a lista completa de chaves está em `helm show values`.

Detalhes de configuração

Personalização de propriedades do NiFi

É possível personalizar propriedades do NiFi adicionando `nifiProperties.overrideConfigs` em `nifiCluster.readOnlyConfig`. Isso permite ajustar persistência, desempenho e outras configurações específicas do NiFi.

Exemplo:

```
nifiCluster:
  readOnlyConfig:
    nifiProperties:
      overrideConfigs: |
        nifi.flowfile.repository.checkpoint.interval=5 mins
```

Casos de uso comuns:

- Ajustar intervalos de checkpoint
- Configurar implementação do repositório de provenênça
- Ajustar configurações do repositório de conteúdo
- Reduzir E/S de disco em ambientes de desenvolvimento

© Copyright Tecnisys 2026. All rights reserved.



> **NOTE**
>
> Algumas propriedades podem afetar a persistência de dados ou o desempenho. Revise a documentação do NiFi antes de aplicar em ambientes de produção.

Acesso & Segurança

Acesso

Por padrão, o chart expõe a UI do NiFi por um Service do tipo `NodePort`, normalmente chamado `tdp-nifi-service`. Esse acesso é útil para validação inicial, laboratórios ou ambientes em que o tráfego direto por IP de nó é permitido.

Terminal input

```
kubectl -n <NAMESPACE> get svc tdp-nifi-service
kubectl get nodes -o wide
```

A URL segue o formato:

```
http://<NODE_IP>:<NODE_PORT>/nifi
```

Port-forward

Terminal input

```
kubectl port-forward svc/tdp-nifi-service 8080:8080 -n <NAMESPACE>
```

Abra `http://localhost:8080/nifi`.

Exposição externa

Para exposição externa por Ingress ou Gateway API, consulte Exposição externa — NiFi. Essa página concentra os exemplos completos para Kubernetes padrão, OpenShift, HTTPS e Gateway API.

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Quando a UI não abre mas os *Pods* parecem `Running`, verifique primeiro o `NifiCluster` (`describe`) e eventos do *namespace*: o operador costuma registrar falhas de *listener*, TLS ou ZooKeeper antes que o serviço HTTP responda. Comandos abaixo ajudam a correlacionar fase do CRD, eventos e estado dos *workloads`.

Terminal input

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get nificlusters
kubectl -n <NAMESPACE> describe nificluster <CLUSTER_NAME>
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
kubectl -n <NAMESPACE> logs <POD_NAME> -f
```


Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

HOOK DE PRÉ-EXCLUSÃO

O chart inclui um hook `pre-delete` que remove o CR `NifiCluster` e seus finalizadores antes de o Helm desinstalar o operador. Isso evita que o CR fique preso em estado `Terminating`.

PVCS EM PRODUÇÃO

Para evitar perda de dados ao desinstalar em produção, defina `reclaimPolicy: Retain` nos PVCs de dados:

© Copyright Tecnisys 2026. All rights reserved.



```
nifiCluster:
  nodeConfigGroups:
    default_group:
      storageConfigs:
        - name: flowfile-repository
          reclaimPolicy: Retain
        - name: content-repository
          reclaimPolicy: Retain
        - name: provenance-repository
          reclaimPolicy: Retain
```

© Copyright Tecnisys 2026. All rights reserved.

NiFi – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 1.28.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

O chart `tdp-nifi` suporta três modos operacionais de segurança, controlados pelos blocos `nifiCluster.security` e `nifiCluster.authorizersSecret`. A escolha do modo determina se o NiFi opera em HTTP ou HTTPS, se exige login LDAP e quem gerencia as políticas de acesso.

Modos de segurança

Modo	Quando usar	Autenticação	Autorização
Modo 1	Laboratórios locais e testes funcionais temporários	Sem login	Sem controle por usuário
Modo 2	Ambientes com usuários LDAP e políticas locais no NiFi	LDAP	Arquivo local do NiFi (`file`)
Modo 3	Ambientes com LDAP e políticas centralizadas no Ranger	LDAP	Apache Ranger (`ranger`)

Modo 1: sem autenticação e sem controle de políticas

Use este modo em laboratórios locais ou testes funcionais temporários.

© Copyright Tecnisys 2026. All rights reserved.

- NiFi opera em HTTP
- LDAP desabilitado
- sem tela de login
- acesso não segmentado por usuário/perfil

```
nifiCluster:
  security:
    enabled: false
    protocol: "http"
    port: 8080

    ldap:
      enabled: false

    authorization:
      mode: "file"
```

Se o Ingress estiver habilitado, use o protocolo HTTP no backend:

```
ingress:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
  rules:
    - host: nifi.local
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: tdp-nifi-service
              port:
                number: 8080
```

Modo 2: LDAP com políticas gerenciadas pelo NiFi

Use este modo quando os usuários autenticam via LDAP e as políticas de acesso são geridas pelo próprio NiFi (autorizador local).

- NiFi opera em HTTPS
- login via LDAP

© Copyright Tecnisys 2026. All rights reserved.



- políticas controladas pelo NiFi usando o autorizador de arquivo local (`file`)

```
nifiCluster:
  security:
    enabled: true
    protocol: "https"
    port: 8443

    ldap:
      enabled: true
      url: "ldap://<LDAP_HOST>:389"
      searchBase: "<LDAP_USER_BASE_DN>"
      searchFilter: "(&(uid={0})(objectClass=inetOrgPerson))"
      managerDn: "<LDAP_BIND_DN>"
      managerPassword: "<LDAP_BIND_PASSWORD>"
      referralStrategy: "FOLLOW"
      identityStrategy: "USE_USERNAME"

    authorization:
      mode: "file"
```

Para o primeiro acesso, defina as identidades iniciais via `nifiCluster.authorizersSecret`:

```
nifiCluster:
  authorizersSecret:
    enabled: true
    initialUserIdentities:
      - fry
      - leela
    initialAdminIdentity: fry
```

> **IDENTIDADES NO AUTHORIZERSSECRET**
>
> - `initialAdminIdentity` é o primeiro administrador do NiFi.
> - A identidade deve corresponder ao que o LDAP retorna após a estratégia de mapeamento configurada.
> - Com `identityStrategy: USE_USERNAME`, use identidades como `fry`, `leela`, `admin`.

© Copyright Tecnisys 2026. All rights reserved.

Se o Ingress estiver habilitado, use o protocolo HTTPS no backend:

```
ingress:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
  rules:
    - host: nifi.local
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: tdp-nifi-service
              port:
                number: 8443
```

Gerenciamento declarativo de usuários (opcional)

Após o bootstrap, é possível gerenciar usuários e grupos de forma declarativa via NiFiKop:

```
nifiCluster:
  authorization:
    enabled: true
    managedAdminUsers:
      - name: fry-admin
        identity: fry
    managedReaderUsers:
      - name: hermes-reader
        identity: hermes
```

Use quando quiser repeatibilidade via Helm. Se preferir, faça o bootstrap com `authorizersSecret` e gerencie as políticas depois na UI do NiFi.

Modo 3: LDAP com políticas gerenciadas pelo Ranger

Use este modo quando os usuários autenticam via LDAP e as políticas de acesso são controladas pelo Apache Ranger.

Requisitos:

© Copyright Tecnisys 2026. All rights reserved.

- imagem NiFi com o plugin Ranger já embutido
- overlay `tdp-ranger` para o fluxo de autorização NiFi
- Secret `ranger-nifi-client-tls-password`

```
nifiCluster:
  clusterImage: "registry.tecnisys.com.br/tdp/images/nifi:1.28.0-0"

  security:
    enabled: true
    protocol: "https"
    port: 8443
    additionalProxyHosts:
      - "nifi-headless.<NAMESPACE>.svc.cluster.local"
      - "nifi-headless.<NAMESPACE>.svc.cluster.local:8443"

    ldap:
      enabled: true
      url: "ldap://<LDAP_HOST>:389"
      searchBase: "<LDAP_USER_BASE_DN>"
      searchFilter: "(&(uid={0})(objectClass=inetOrgPerson))"
      managerDn: "<LDAP_BIND_DN>"
      managerPassword: "<LDAP_BIND_PASSWORD>"
      referralStrategy: "FOLLOW"
      identityStrategy: "USE_USERNAME"

    authorization:
      mode: "ranger"
```

Crie o Secret com a senha JKS do Ranger antes do deploy:

Terminal input

```
kubectl -n <NAMESPACE> create secret generic ranger-nifi-client-tls-password \
  --from-literal=password='<KEYSTORE_PASSWORD>' \
  --dry-run=client -o yaml | kubectl apply -f -
```

Aplique os overlays correspondentes:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install tdp-ranger oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

Terminal input

```
helm upgrade --install tdp-nifi oci://registry.tecnisys.com.br/tdp/charts/tdp-nifi \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

REGRAS OPERACIONAIS DO MODO 3

- Use a imagem NiFi customizada com o plugin Ranger já incluído.
- `nifiCluster.security.authorization.ranger.adminIdentity` deve corresponder à identidade do certificado cliente Ranger (padrão no repositório TDP: `CN=ranger-nifi-client`).
- O serviço Ranger deve apontar para `https://<NIFI_SERVICE>/nifi-api/resources`.
- Para NiFi em cluster, o Ranger precisa de políticas técnicas para `/resources`, `/controller`, `/system` e `/proxy`.

Resumo: chaves por modo

Configuração	Modo 1	Modo 2	Modo 3
`nifiCluster.security.enabled`	`false`	`true`	`true`
`nifiCluster.security.protocol`	`http`	`https`	`https`
`nifiCluster.security.ldap.enabled`	`false`	`true`	`true`

© Copyright Tecnisys 2026. All rights reserved.

Configuração	Modo 1	Modo 2	Modo 3
`nifiCluster.security.authorization.mode`	`file`	`file`	`ranger`
`nifiCluster.authorizersSecret`	não relevante	bootstrap obrigatório	não é fonte de políticas
`nifiCluster.authorization`	opcional	opcional	não é a fonte principal
Configuração de autorização Ranger	não	não	sim
imagem NiFi com plugin Ranger	não	não	sim

Regras práticas

- Para **sem login**: desabilite `security.enabled` e `ldap.enabled`.
- Para **login via LDAP com políticas locais no NiFi**: mantenha `authorization.mode: file`.
- Para **login via LDAP com políticas no Ranger**: use `authorization.mode: ranger`.

Configuração LDAP detalhada

O bloco `nifiCluster.security.ldap` controla todos os parâmetros de busca e autenticação LDAP:

Campo	Descrição
`url`	Endereço do servidor LDAP

© Copyright Tecnisys 2026. All rights reserved.

Campo	Descrição
`searchBase`	Base DN onde os usuários estão localizados
`searchFilter`	Filtro de busca — `{0}` é substituído pelo username informado
`authenticationStrategy`	Tipo de bind: `SIMPLE` para autenticação direta
`managerDn`	DN do usuário de bind (somente leitura no diretório)
`managerPassword`	Senha do usuário de bind — não versionar em repositório público
`referralStrategy`	Como seguir referências LDAP: `FOLLOW` ou `IGNORE`
`identityStrategy`	`USE_USERNAME` usa o login como identidade no NiFi

> **CREDENCIAIS DE BIND**
>
> Não armazene `managerPassword` em repositório Git. Use um arquivo de valores privado (fora do controle de versão) ou um mecanismo de gestão de Secrets para injetar essa credencial no deploy.

Para LDAPS (TLS), substitua `ldap://` por `ldaps://` e use a porta 636.

Certificados TLS (cert-manager)

O NiFi opera em HTTPS com certificados gerados automaticamente via `cert-manager`. A configuração SSL fica sob `nifiCluster.security.ssl`:

© Copyright Tecnisys 2026. All rights reserved.

```
nifiCluster:
  security:
    ssl:
      enabled: true
      secrets:
        create: true
        tlsSecretName: "nifi-cluster-ca"
        pkiBackend: cert-manager
        issuerRef:
          name: selfsigned-issuer
          kind: Issuer
```

Campo	Descrição
`ssl.enabled`	Habilita TLS (derivado automaticamente de `protocol: https`)
`ssl.secrets.create`	Cria automaticamente o Secret de TLS
`ssl.secrets.pkiBackend`	`cert-manager` delega a emissão ao cert-manager
`ssl.secrets.issuerRef.name`	Nome do Issuer ou ClusterIssuer no cluster

Pré-requisito: cert-manager via tdp-crds

No TDP, o chart `tdp-crds` instala os CRDs necessários, incluindo os do cert-manager. Instale o `tdp-crds` antes do `tdp-nifi`:

Terminal input

```
helm upgrade --install tdp-crds \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  --namespace tdp --create-namespace
```

© Copyright Tecnisys 2026. All rights reserved.



Quando `nifikop.webhook.enabled=true`, os controllers do cert-manager (`cert-manager`, `cainjector` e `webhook`) também precisam estar em execução no cluster.

Boas práticas

Aspecto	Recomendação
Ambiente de desenvolvimento	Modo 1 (sem autenticação), acesso via port-forward
Ambiente compartilhado / produção	Modo 2 com LDAPS (porta 636) e HTTPS habilitado
Credenciais	Nunca versionar `managerPassword` em valores públicos
Certificados	Usar `cert-manager` via `tdp-crds` em produção
Políticas centralizadas	Modo 3 com Ranger quando houver auditoria e controle centralizado de acesso

Solução de problemas

Problema	Causa provável	Solução
Pod NiFi não sobe com HTTPS	`cert-manager` ausente ou Issuer não encontrado	Verificar instalação do cert-manager e nome do Issuer
Login LDAP falha	DN de bind ou senha incorretos	Testar bind com `ldapsearch` antes de configurar
Certificado inválido no navegador	Uso de `selfsigned-issuer`	Esperado em desenvolvimento; usar CA válida em produção

© Copyright Tecnisys 2026. All rights reserved.

Problema	Causa provável	Solução
Plugin Ranger não conecta	`adminIdentity` incorreto ou Secret JKS ausente	Verificar `nifiCluster.security.authorization.ranger.adminIdentity` e Secret `ranger-nifi-client-tls-password`
NifiCluster preso em estado intermediário	Webhook do NiFiKop sem cert-manager	Verificar `nifikop.webhook.enabled` e se os controllers do cert-manager estão prontos

© Copyright Tecnisys 2026. All rights reserved.

NiFi – Integração

CHART

Version 3.0.1 | Type application | AppVersion 1.28.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O Apache NiFi pode ser integrado ao **Apache Ranger** para controle centralizado de acesso a flows e componentes. Essa integração é configurada pelo chart `tdp-ranger` (não pelo `tdp-nifi`): o Ranger instala um plugin no NiFi e passa a ser a fonte de autoridade para todas as decisões de acesso.

Essa página descreve o registro e a configuração da integração pelo lado do Ranger. Para configurar o NiFi para usar o Ranger como mecanismo de autorização (`nifiCluster.security.authorization.mode: "ranger"`), consulte Segurança — NiFi.

Integração com Apache Ranger

Como funciona

O plugin Ranger para NiFi intercepta as solicitações de acesso aos components do NiFi (ProcessGroups, Processors, portas de entrada/saída) e as valida contra as políticas definidas no Ranger Admin. Políticas padrão são criadas automaticamente pelo `configJob` do chart `tdp-ranger` quando a integração está habilitada.

Habilitar a integração

A configuração é feita no chart `tdp-ranger`, não no `tdp-nifi`. O chart Ranger precisa ser instalado com `rangerIntegrations.nifi.enabled=true` e saber onde o NiFi está rodando:

© Copyright Tecnisys 2026. All rights reserved.

```
rangerIntegrations:
  configJob:
    enabled: true
  nifi:
    enabled: true
    serviceName: "<NIFI_SERVICE>"
    serviceDisplayName: "NiFi TDP"
    connection:
      nifiUrl: "http://<NIFI_SERVICE>.<NAMESPACE>.svc.cluster.local:8080"
      authenticationType: "none"
    credentials:
      username: "<USERNAME>"
      password: "<PASSWORD>"
```

> **PRÉ-REQUISITO**
>
> O NiFi deve estar implantado e acessível a partir do namespace do Ranger antes de habilitar essa integração. O `configJob` aguarda o Ranger ficar pronto, mas o NiFi precisa estar rodando quando o job executar o registro.

Instalação ou atualização com a integração

Terminal input

```
helm upgrade --install <RANGER_RELEASE> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  -f <BASE_VALUES_FILE> \
  -f <VALUES_FILE>
```

Ou via `--set`:

Terminal input

```
helm upgrade --install <RANGER_RELEASE> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  --set rangerIntegrations.nifi.enabled=true \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  --set rangerIntegrations.nifi.connection.nifiUrl=http://<NIFI_SERVICE>.
<NAMESPACE>.svc.cluster.local:8080
```

Verificação

Após a integração, verifique no Ranger Admin que o serviço NiFi aparece em **Service Manager**:

Terminal input

```
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<RANGER_RELEASE>-ranger-integrations
```

No Ranger Admin, acesse **NiFi** → **Policies** para ver as políticas criadas automaticamente.

Parâmetros principais

Parâmetro	Descrição
`rangerIntegrations.nifi.enabled`	Habilitar a integração NiFi no Ranger
`rangerIntegrations.nifi.serviceName`	Nome do serviço no Ranger
`rangerIntegrations.nifi.connection.nifiUrl`	URL do NiFi acessível do cluster
`rangerIntegrations.nifi.connection.authenticationType`	Tipo de autenticação (`none` para acesso sem credenciais)
`rangerIntegrations.nifi.credentials.username`	Usuário admin do NiFi
`rangerIntegrations.nifi.credentials.password`	Senha do admin do NiFi

Consulte `helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger` para a lista completa de parâmetros da integração NiFi.

© Copyright Tecnisys 2026. All rights reserved.

OpenMetadata

CHART

Version 3.0.1 | Type application | AppVersion 1.12.4

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | External DB suportado | Ingress suportado | Gateway API suportado

Esta página cobre a instalação e configuração do **OpenMetadata** via chart `tdp-openmetadata`: pré-requisitos, instalação OCI, parâmetros principais, acesso e desinstalação. Autenticação, LDAP e job de datasources são detalhados nas páginas relacionadas. O banco de dados (PostgreSQL) está documentado nesta página.

O que é o OpenMetadata?

O OpenMetadata é uma plataforma de **catálogo de dados e governança**.

Em termos práticos, é onde você registra todos os ativos de dados da sua organização — tabelas do ClickHouse, tópicos do Kafka, dashboards do Superset, pipelines do Airflow — e obtém uma visão unificada de quem os criou, o que significam, quem os usa e como eles se relacionam.

No TDP Kubernetes, o OpenMetadata resolve a seguinte necessidade: com tantos componentes (Trino, ClickHouse, Kafka, Airflow, Superset), fica difícil saber onde um dado específico está, o que significa, e se ainda é confiável.

O OpenMetadata centraliza a descoberta, a catalogação e a documentação dos ativos de dados.
Isso facilita a descoberta da origem, do contexto e dos responsáveis pelos dados em ambientes com múltiplos motores.
O catálogo não substitui o monitoramento de pipelines ou SLAs, mas organiza metadados e vínculos entre ativos registrados na plataforma.

© Copyright Tecnisys 2026. All rights reserved.



> **PARA SABER MAIS**
>
> Consulte OpenMetadata — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

Os valores do servidor OpenMetadata ficam sob a chave `tdp-openmetadata:`. Configurações de serviços externos, automações e exposição externa ficam sob blocos separados no nível raiz:

- `tdp-openmetadata:` — servidor, imagem, Ingress, banco de dados, backend de busca, RBAC.
- `externalServices:` — referência ao PostgreSQL externo (hooks e senha admin).
- `datasourcesIntegration:` — job pós-instalação de registro automático de datasources TDP.
- `gatewayApi:` — configuração do HTTPRoute e Gateway para exposição via Gateway API.
- `TDP-Settings:` — seleção do modo de exposição HTTP/HTTPS (Ingress ou Gateway API).
- `openshift:` — configuração de SCC para ambientes OpenShift.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false
    gatewayApi:
      enabled: false

tdp-openmetadata:
  openmetadata:
    config:
      database:
        dbScheme: "mysql"

externalServices:
  postgresql:
    enabled: false
```

© Copyright Tecnisys 2026. All rights reserved.



> **CHAVE DE CONTROLE DE EXPOSIÇÃO**
>
> Este chart usa a chave `TDP-Settings.gateway` para controlar qual modo de exposição HTTP está ativo. Habilite apenas um: Ingress **ou** Gateway API.

Componentes implantados

Componente	Descrição
OpenMetadata Server	API e interface web do catálogo
MySQL (interno padrão)	Banco de metadados do OpenMetadata
OpenSearch	Backend de busca para encontrar ativos de dados
K8s Jobs de ingestão	Executor de pipelines de ingestão (padrão; sem dependência de Airflow)
Job de datasources (opcional)	Registra automaticamente os serviços TDP após a instalação

Automações do chart

O chart `tdp-openmetadata` inclui as seguintes automações:

- **URLs dinâmicas** — endpoints de API e JWKS são derivados automaticamente do nome do release e do namespace, sem necessidade de configuração manual.
- **Bootstrap de credenciais** — o usuário admin (`admin@open-metadata.org`) é criado automaticamente; a senha é configurável via values e exibida na saída do `helm install` (NOTES).
- **Secrets gerados automaticamente** — dependências empacotadas (MySQL, OpenSearch) recebem secrets Kubernetes gerados pelo próprio chart.
- **Limpeza no uninstall** — um Job `post-delete` remove automaticamente os PVCs, secrets e configmaps deixados pelo release ao executar `helm uninstall`.

Visão geral

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Chart	`tdp-openmetadata`
Versão do OpenMetadata	1.12.4
Versão do Chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata`

Páginas relacionadas

Página	Conteúdo
Integrações — OpenMetadata	Job de registro de datasources, arquivos de valores modulares, configuração por serviço e execução de pipelines
Segurança — OpenMetadata	Autenticação básica e via LDAP, controle de acesso e boas práticas
Exposição externa	Ingress e Gateway API para o OpenMetadata

Pré-requisitos

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível no cluster para volumes persistentes.

Instalação

Instalação (OCI)

Instalação no Kubernetes

© Copyright Tecnisys 2026. All rights reserved.



O comando a seguir cria o release no namespace indicado. Use `--wait --wait-for-jobs` para aguardar a conclusão dos Jobs pós-instalação antes de retornar — sem esses flags o helm pode reportar sucesso antes de os Jobs terminarem.

Terminal input

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
openmetadata \
  -n <NAMESPACE> --create-namespace \
  --wait --wait-for-jobs --timeout 10m
```

Instalação no OpenShift

No OpenShift, os Jobs de ingestão executam com `runAsUser: 0` (root) para instalar dependências via pip. Por isso requerem a SCC (*Security Context Constraint*) `anyuid`. Defina `openshift.enabled: true` para que o chart configure automaticamente o RoleBinding necessário.

Terminal input

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
openmetadata \
  -n <NAMESPACE> --create-namespace \
  --set openshift.enabled=true \
  --wait --wait-for-jobs --timeout 10m
```

Ou via arquivo de valores:

values-openshift.yaml

```
openshift:
  enabled: true
```

Terminal input

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
openmetadata \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  -n <NAMESPACE> --create-namespace \
  -f values-openshift.yaml \
  --wait --wait-for-jobs --timeout 10m
```

Ao habilitar `openshift.enabled: true`, o chart cria um RoleBinding que adiciona as seguintes ServiceAccounts à SCC `anyuid`:

ServiceAccount	Uso
`openmetadata-ingestion`	Jobs de ingestão do OpenMetadata
`<RELEASE_NAME>-openmetadata-sa`	Job de integração de datasources

Após criar um serviço de dados na UI do OpenMetadata, verifique que os Jobs iniciaram corretamente:

Terminal input

```
oc get jobs -n <NAMESPACE>
oc get pods -n <NAMESPACE>
```

Parâmetros principais

Parâmetros principais

A tabela abaixo destaca chaves que definem **se o stack sobe**, **qual imagem** roda, **como o servidor alcança MySQL e OpenSearch**, se há **Ingress**, e se entram em cena **PostgreSQL externo** ou o job de datasources. Para a lista completa de parâmetros, consulte `helm show values` na versão do chart instalada.

Parâmetro	Descrição	Valor padrão
`tdp-openmetadata.enabled`	Habilitar o deploy do OpenMetadata	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`tdp-openmetadata.image.tag`	Tag da imagem do servidor	1.12.4
`TDP-Settings.gateway.ingress.enabled`	Habilitar Ingress para exposição externa	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilitar Gateway API para exposição externa	`false`
`tdp-openmetadata.openmetadata.config.database.dbScheme`	Esquema do banco de metadados	`mysql`
`tdp-openmetadata.openmetadata.config.database.host`	Host do banco de metadados	`tdp-openmetadata-mysql`
`tdp-openmetadata.openmetadata.config.elasticsearch.searchType`	Backend de busca	`opensearch`
`tdp-openmetadata.openmetadata.config.elasticsearch.host`	Host do backend de busca	`tdp-openmetadata-opensearch`
`tdp-openmetadata.openmetadata.config.pipelineServiceClientConfig.type`	Modo de execução de pipelines: `k8s` ou `airflow`	`k8s`
`externalServices.postgresql.enabled`	Usar PostgreSQL externo em vez do MySQL empacotado	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`datasourcesIntegration.enabled`	Job pós-install/upgrade que registra serviços externos	`false`
`openshift.enabled`	Configurar SCC `anyuid` para ambientes OpenShift	`false`

Os blocos individuais de cada datasource ficam desabilitados por padrão mesmo quando o job está ativo. Habilite apenas os serviços cujos dados de conexão e credenciais estiverem configurados.

Demais chaves: consulte a saída de `helm show values` para a versão do pacote que utiliza.

Detalhes de configuração

Integração

O OpenMetadata tem um papel diferente dos demais componentes TDP. Enquanto Ozone, ClickHouse e Trino são **consumidos** por outras ferramentas (que leem ou escrevem dados neles), o OpenMetadata é quem **cataloga e governa** os outros: ele descobre, documenta e conecta os ativos de dados de toda a plataforma.

Isso significa que a integração do OpenMetadata é, na prática, **conectar o catálogo a cada um dos outros componentes da stack**:

Componente registrado	Tipo de serviço no catálogo
Trino	Database Service
ClickHouse	Database Service
Superset	Dashboard Service

© Copyright Tecnisys 2026. All rights reserved.

Componente registrado	Tipo de serviço no catálogo
Airflow	Pipeline Service
Kafka	Messaging Service

O chart `tdp-openmetadata` automatiza esse registro por meio de um **Job pós-install/upgrade** (`datasourcesIntegration`), que autentica na API do OpenMetadata e cria os serviços habilitados. Você escolhe quais componentes registrar — cada um tem seu próprio bloco de configuração e pode ser habilitado ou desabilitado individualmente.

Além do registro de datasources, o chart também suporta **substituição do MySQL interno por PostgreSQL externo** (`tdp-postgresql`). Consulte a seção Banco de dados (PostgreSQL) nesta página.

Para arquivos de valores modulares, configuração por serviço, exemplos de desenvolvimento e produção, verificação e execução de pipelines, consulte Integrações — OpenMetadata.

Execução de pipelines

O OpenMetadata pode executar pipelines de ingestão em dois modos: `k8s` (padrão), com Jobs Kubernetes, ou `airflow`, com integração REST com o serviço Airflow.

O comportamento é controlado por `pipelineServiceClientConfig.type`. Parâmetros por modo e requisitos do Airflow estão em Integrações — OpenMetadata.

Banco de dados (PostgreSQL)

Por padrão, o OpenMetadata usa um MySQL empacotado para armazenar seus metadados. Esta seção descreve como substituí-lo por um PostgreSQL gerenciado pelo chart `tdp-postgresql`.

O que acontece ao habilitar

Ao definir `externalServices.postgresql.enabled: true`, o chart executa automaticamente a seguinte sequência:

© Copyright Tecnisys 2026. All rights reserved.



1. **MySQL desabilitado** — nenhum pod MySQL é criado; PVCs MySQL não são provisionados.
2. **Senha gerada automaticamente** — uma senha aleatória é criada para o usuário `openmetadata_user` e salva em um Kubernetes Secret chamado `openmetadata-postgresql-secrets`.
3. **Job `pre-install`/`pre-upgrade`** — um Job cria o banco `openmetadata_db` e o usuário `openmetadata_user` no PostgreSQL, usando a senha de admin do `tdp-postgresql` (Secret gerado automaticamente por aquele chart ao instalar).
4. **OpenMetadata conecta ao PostgreSQL** — após o Job concluir, o servidor OpenMetadata inicia e conecta ao PostgreSQL usando as credenciais geradas no passo anterior.

Os Secrets envolvidos nesta integração são:

Secret	Criado por	Papel
`<POSTGRESQL_RELEASE>-postgresql`	Chart `tdp-postgresql` (pré-existente)	Senha do admin do PostgreSQL — lida pelo Job de criação de banco
`openmetadata-postgresql-secrets`	Chart `tdp-openmetadata` (hook weight -10)	Senha do usuário `openmetadata_user` — usada pelo OpenMetadata Server
`<RELEASE_NAME>-postgresql-secrets`	Chart `tdp-openmetadata` (hook weight -10)	Cópia prefixada pelo nome do release do secret anterior

Detalhe dos hooks

Quando `externalServices.postgresql.enabled: true`, o chart executa os seguintes hooks antes de subir o servidor:

1. **Hook `pre-install`/`pre-upgrade` (weight -10)** — cria dois Secrets:
   - `openmetadata-postgresql-secrets` (usado pelo OpenMetadata)
   - `<RELEASE_NAME>-postgresql-secrets` (cópia prefixada pelo nome do release)
2. **Hook `pre-install`/`pre-upgrade` (weight -5)** — Job de criação de banco:

© Copyright Tecnisys 2026. All rights reserved.



- conecta ao PostgreSQL usando a senha de admin do Secret do `tdp-postgresql`
- cria o banco `openmetadata_db` e o usuário `openmetadata_user`
- define o usuário como owner do banco

3. **Recursos regulares** — o servidor OpenMetadata inicia e conecta ao PostgreSQL.

Pré-requisitos

- Chart `tdp-postgresql` implantado no mesmo namespace (ou namespace acessível).
- Secret de admin do PostgreSQL criado automaticamente pelo `tdp-postgresql`.

Quick start

Adicione a configuração do PostgreSQL ao seu arquivo de values e aplique via Helm:

```
externalServices:
  postgresql:
    enabled: true
    releaseName: "tdp-postgresql"

openmetadata-dependencies:
  mysql:
    enabled: false

tdp-openmetadata:
  openmetadata:
    config:
      database:
        enabled: true
        host: "tdp-postgresql"
        port: 5432
        driverClass: org.postgresql.Driver
        dbScheme: postgresql
        databaseName: openmetadata_db
        auth:
          username: openmetadata_user
          password:
            secretRef: "openmetadata-postgresql-secrets"
            secretKey: "openmetadata-postgresql-password"
```

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE> \
  --wait --wait-for-jobs --timeout 20m
```

Configuração manual

Use esta opção quando preferir controle total sobre os parâmetros ou quando não quiser depender do arquivo de overlay fornecido. É necessário configurar explicitamente três blocos: habilitar o PostgreSQL externo, desabilitar o MySQL interno e apontar o OpenMetadata para o novo banco.

Configure os três blocos abaixo no arquivo de valores:

```
# 1. Habilitar PostgreSQL externo
externalServices:
  postgresql:
    enabled: true
    releaseName: "tdp-postgresql"  # nome do release tdp-postgresql
    namespace: ""                   # vazio = mesmo namespace do OpenMetadata
    database:
      secretName: "openmetadata-postgresql-secrets"  # gerado automaticamente

# 2. Desabilitar MySQL interno
openmetadata-dependencies:
  mysql:
    enabled: false

# 3. Configurar OpenMetadata para PostgreSQL
tdp-openmetadata:
  openmetadata:
    config:
      database:
        enabled: true
        host: "tdp-postgresql"
        port: 5432
        driverClass: org.postgresql.Driver
        dbScheme: postgresql
        databaseName: openmetadata_db
        auth:
          username: openmetadata_user
```

© Copyright Tecnisys 2026. All rights reserved.



```
          password:
            secretRef: "openmetadata-postgresql-secrets"
            secretKey: "openmetadata-postgresql-password"
```

Deploy em namespaces distintos

Quando o `tdp-postgresql` está instalado em um namespace diferente do OpenMetadata — por exemplo, em uma instalação compartilhada de banco de dados — é necessário informar o namespace no bloco `externalServices` e usar o FQDN completo do serviço no campo `host`, para que o OpenMetadata alcance o PostgreSQL entre namespaces.

Se o PostgreSQL estiver em outro namespace, use o FQDN do serviço:

```
externalServices:
  postgresql:
    enabled: true
    releaseName: "tdp-postgresql"
    namespace: "postgresql-namespace"

tdp-openmetadata:
  openmetadata:
    config:
      database:
        host: "tdp-postgresql.postgresql-namespace.svc.cluster.local"
```

Verificação

Terminal input

```
# Verificar Job de criação do banco
kubectl -n <NAMESPACE> get jobs -l app.kubernetes.io/instance=<RELEASE_NAME>

# Verificar o secret gerado
kubectl -n <NAMESPACE> get secret openmetadata-postgresql-secrets -o yaml

# Confirmar o banco no PostgreSQL
kubectl exec -n <NAMESPACE> <POSTGRESQL_POD> -- psql -U postgres -c "\l
openmetadata_db"
```

© Copyright Tecnisys 2026. All rights reserved.

EXEMPLO

Exemplo

```
kubectl exec -n <NAMESPACE> tdp-postgresql-0 -- psql -U postgres -c "\l
openmetadata_db"
```

Resolução de problemas

Sintoma	Causa provável	Correção
Job falha com "secret not found"	Secret de admin do PostgreSQL ausente	Confirme que `tdp-postgresql` está implantado e o secret existe
Job falha com "connection refused"	Nome do serviço ou namespace incorreto	Verifique `host` e `namespace` nos values
OpenMetadata não conecta	Usuário não criado	Verifique logs do `db-create-job`
"Secret already exists"	Secret criado manualmente	Remova o secret manual e deixe o chart gerenciar

Migração do MySQL para PostgreSQL

1. Faça backup dos dados (os dados do MySQL **não são migrados automaticamente**).
2. Desinstale o release atual (mantenha PVCs se quiser referência).
3. Reinstale com o overlay PostgreSQL (instalação limpa com PostgreSQL).

Acesso & Segurança

Exposição externa

Ingress e Gateway API são mutuamente exclusivos, controlados por `TDP-Settings.gateway.*`. No OpenMetadata, o Ingress fica sob `tdp-openmetadata.ingress.*`

© Copyright Tecnisys 2026. All rights reserved.

e o Gateway API sob `gatewayApi.openmetadata.*` (porta 8585).

Para os exemplos completos de values (Ingress e Gateway API), consulte Exposição externa.

Acesso

Enquanto o Ingress permanecer desabilitado ou em configuração, utilize o port-forward abaixo para acessar a interface e a API na porta padrão do serviço.

Terminal input

```
kubectl port-forward svc/<RELEASE_NAME> 8585:8585 -n <NAMESPACE>
```

Acesse `http://localhost:8585` no navegador. As credenciais padrão são exibidas na saída do `helm install` (NOTES):

- **Usuário:** `admin@open-metadata.org`
- **Senha:** `admin` (padrão; altere em produção via values)

Segurança

O OpenMetadata suporta dois modos de autenticação:

- **Básico** (padrão): usuário/senha local gerenciado pelo próprio OpenMetadata.
- **LDAP**: autenticação delegada a um diretório corporativo (AD, OpenLDAP etc.).

Em produção, desabilite o auto-cadastro (`enableSelfSignup: false`) para controlar quem pode criar contas. As credenciais do administrador inicial e do job de datasources não devem ser armazenadas em texto simples nos values — use Kubernetes Secrets.

Para configuração detalhada — LDAP (DNs, atributos, roles, truststore), controle de acesso, Secrets e boas práticas operacionais — consulte Segurança — OpenMetadata.

Solução de problemas

Solução de problemas

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<RELEASE_NAME>-register-datasources
```

Falhas comuns: OpenMetadata ainda não pronto, erro de autenticação, Secret ausente ou serviço de destino inacessível. Para diagnóstico detalhado, consulte Integrações — OpenMetadata.

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

O chart inclui um Job `post-delete` que remove automaticamente os recursos deixados pelo release:

- **PVCs:** `data-mysql-0`, `opensearch-opensearch-0` e demais volumes do release
- **Secrets:** `mysql-secrets`, `airflow-secrets`, `om-secret-*` (criados pelos Jobs de ingestão)
- **ConfigMaps:** `mysql-init-scripts` e afins
- **Jobs e CronJobs:** `om-job-*`, `om-cronjob-*`, `cron-deploy-pipelines`, `cron-reindex`
- **Pods concluídos:** `om-job-*`, `om-automation-test-connection-*`

Para preservar PVCs ou secrets para análise, use `--no-hooks` (o Job de limpeza não será executado):

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE> --no-hooks
```

Verificação após a desinstalação:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.



```
kubectl -n <NAMESPACE> get jobs,cronjobs,pods,secrets | \
  grep -E 'om-job-|om-cronjob-|om-secret-|cron-deploy-pipelines|cron-reindex|om-automation-test-connection' || true
```

© Copyright Tecnisys 2026. All rights reserved.

OpenMetadata – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 1.12.4

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

> **PADRÕES GERAIS DE SEGURANÇA**
>
> Para conceitos transversais a todos os componentes TDP — como gerenciamento de Secrets, rotação de credenciais, NetworkPolicy, rate limiting e auditoria — consulte Segurança — Padrões Gerais.

Configuração de Autenticação

Os parâmetros abaixo controlam o modo de autenticação, o auto-cadastro de usuários e a configuração do administrador inicial.

Parâmetro	Descrição	Valor padrão
`tdp-openmetadata.openmetadata.config.authentication.provider`	Modo de autenticação: `basic` (usuário/senha local) ou `ldap`	`basic`
`tdp-openmetadata.openmetadata.config.authentication.enableSelfSignup`	Permite que novos usuários se cadastrem pela UI	`true`
`tdp-openmetadata.openmetadata.config.authorizer.initialAdmins`	Lista de usuários administradores criados no bootstrap	`["admin"]`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`tdp-openmetadata.openmetadata.config.authorizer.principalDomain`	Domínio adicionado ao nome dos `initialAdmins`	`open-metadata.org`
`datasourcesIntegration.openmetadata.auth.*`	Credenciais usadas pelo Job de datasources para autenticar na API; quando `provider=basic`, são também exibidas no NOTES do Helm	`admin@open-metadata.org` / `admin`

Para desabilitar o auto-cadastro em ambientes de produção:

```
tdp-openmetadata:
  openmetadata:
    config:
      authentication:
        enableSelfSignup: false
```

Autenticação via LDAP

O pacote `tdp-openmetadata` suporta autenticação via LDAP, permitindo o uso de diretórios corporativos para login no OpenMetadata.

Para habilitar esse modo, crie um Secret Kubernetes com a senha de bind e adicione a configuração LDAP ao seu arquivo de valores. e crie um Secret Kubernetes com a senha de bind.

A configuração permite customizar DNs de usuários e grupos, atributos de e-mail e login, mapeamento de grupos para roles e o comportamento do truststore.

Após a ativação, o login passa a ser feito com as credenciais LDAP do usuário.

Como habilitar

1. Ative `ldap.enabled: true` no `values.yaml`.

© Copyright Tecnisys 2026. All rights reserved.

2. Inclua as definições de LDAP no seu arquivo de valores.

Exemplo de instalação/atualização com overlay:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> --create-namespace \
  -f <VALUES_FILE>
```

Pré-requisitos

- Servidor LDAP acessível a partir do cluster Kubernetes
- Bind DN com permissões para procurar usuários e grupos
- Secret Kubernetes com a senha de bind

Credenciais e Secrets

Terminal input

```
kubectl create secret generic tdp-openmetadata-ldap-secret \
  --from-literal=bind-password=<PASSWORD> \
  -n <NAMESPACE>
```

Configuração LDAP

Defina no overlay os parâmetros principais do LDAP: host, porta, DN administrativo, base DN de usuários e grupos, além dos atributos e do mapeamento de roles.

<VALUES_FILE>

```
ldap:
  enabled: true

tdp-openmetadata:
  openmetadata:
    config:
      authentication:
```

© Copyright Tecnisys 2026. All rights reserved.

```
        enabled: true
        provider: ldap
        enableSelfSignup: true
        ldapConfiguration:
          host: "ldap.example.local"
          port: 389
          dnAdminPrincipal: "cn=admin,dc=example,dc=local"
          dnAdminPassword:
            secretRef: tdp-openmetadata-ldap-secret   # Secret criado no passo
anterior
            secretKey: bind-password
          userBaseDN: "ou=users,dc=example,dc=local"
          groupBaseDN: "ou=groups,dc=example,dc=local"
          mailAttributeName: mail
          usernameAttributeName: mail
          groupAttributeName: memberOf
          groupMemberAttributeName: member
          maxPoolSize: 3
          authRolesMapping: '{"cn=tdp-admins,ou=groups,dc=example,dc=local":
["Admin"]}'
          authReassignRoles: []
          sslEnabled: false
          trustStoreConfigType: TrustAll   # Use CustomTrustStore ou JVMDefault
para validar certificados
```

> **SENHA DE BIND VIA SECRET**
>
> O campo `dnAdminPassword` deve sempre referenciar um Kubernetes Secret — nunca coloque a senha em texto simples no arquivo de valores. Crie o Secret antes do deploy conforme descrito acima.

O que esta configuração cobre

- Host, porta, DN administrativo e bases de usuários/grupos
- Atributos customizáveis para e-mail, usuário e grupos
- Mapeamento de grupos LDAP para roles do OpenMetadata via `authRolesMapping`
- Truststore padrão `TrustAll`, que não exige LDAPS, mas pode ser ajustado para o modelo do seu ambiente

Login

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl port-forward svc/<RELEASE_NAME> 8585:8585 -n <NAMESPACE>
```

Depois, acesse `http://localhost:8585` e faça login com as credenciais LDAP do usuário.

Para a lista completa de parâmetros, consulte `helm show values` na versão do chart instalada e a documentação oficial do OpenMetadata.

OpenShift: configuração de SCC

No OpenShift, os jobs de ingestão do OpenMetadata executam com `runAsUser: 0` (root) para instalação de pacotes via `pip`. Isso exige o `anyuid` Security Context Constraint (SCC).

Para configurar o SCC automaticamente, ative `openshift.enabled: true` no arquivo de valores:

```
openshift:
  enabled: true
```

Ou via linha de comando:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> --create-namespace \
  --set openshift.enabled=true
```

Quando habilitado, o chart cria um `RoleBinding` que adiciona as seguintes ServiceAccounts ao SCC `anyuid`:

- `openmetadata-ingestion` — usada pelos jobs de ingestão do OpenMetadata
- `<RELEASE_NAME>-openmetadata-sa` — usada pelo job de registro de datasources

© Copyright Tecnisys 2026. All rights reserved.



> **WARNING**
>
> Sem essa configuração no OpenShift, os jobs de ingestão falham ao tentar instalar dependências Python.

Segurança do job de registro de datasources

O job de integração autentica na API do OpenMetadata e acessa serviços externos. Observe as seguintes práticas:

- **Nunca armazene senhas em texto simples no arquivo de valores em produção.** Use `passwordSecret` com referência a um Secret Kubernetes criado separadamente.
- **Restrinja o acesso aos Secrets** via RBAC: o job usa uma ServiceAccount dedicada; não amplie suas permissões além do necessário.
- **Use HTTPS** nos endpoints dos serviços quando estiverem fora do cluster: defina `scheme: "https"` para ClickHouse e Trino, e `verifySSL: "validate"` para Superset e Airflow.
- **Rotacione senhas regularmente**: após trocar a senha de um serviço, atualize o Secret Kubernetes correspondente e re-execute o job de registro.

Exemplo de Secret para credencial do ClickHouse:

Terminal input

```
kubectl create secret generic tdp-clickhouse-tdp-user \
  --from-literal=password=<PASSWORD> \
  -n <NAMESPACE>
```

Referência no arquivo de valores:

```
datasourcesIntegration:
  clickhouse:
    connection:
      passwordSecret:
        name: "tdp-clickhouse-tdp-user"
        key: "password"
```

© Copyright Tecnisys 2026. All rights reserved.

OpenMetadata – Integração

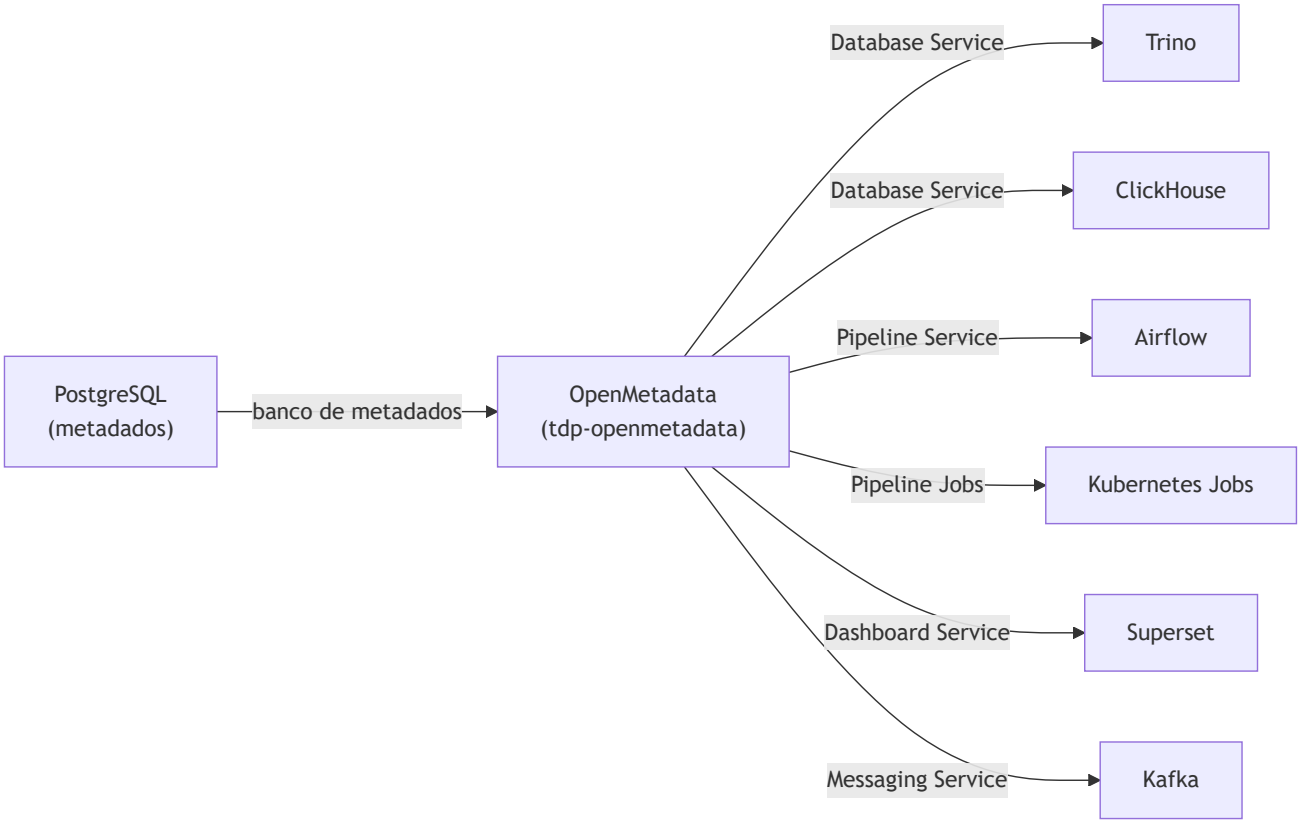
CHART

Version 3.0.1 | Type application | AppVersion 1.12.4

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O chart `tdp-openmetadata` pode usar um PostgreSQL externo (`tdp-postgresql`) como banco de metadados no lugar do MySQL interno — consulte Configuração do OpenMetadata.

O chart `tdp-openmetadata` pode executar um Job pós-*install*/*upgrade* que registra serviços externos no OpenMetadata quando `datasourcesIntegration.enabled=true`. O Job aguarda o OpenMetadata ficar pronto, autentica na API e registra os serviços habilitados.

© Copyright Tecnisys 2026. All rights reserved.



Além disso, o OpenMetadata pode executar pipelines de ingestão em dois modos: por **Kubernetes Jobs** (padrão) ou por **Airflow**.

> **SEGURANÇA NAS INTEGRAÇÕES**
>
> Para práticas de segurança específicas do job de datasources — uso de Secrets Kubernetes, RBAC, HTTPS e rotação de senhas — consulte Segurança — OpenMetadata.

Habilitar integrações por serviço

O job de registro executa durante o install/upgrade quando `datasourcesIntegration.enabled: true`.
Para registrar um serviço no OpenMetadata, habilite o bloco do serviço desejado e preencha os dados de conexão.
Os blocos individuais ficam desabilitados por padrão — apenas o serviço que você habilitar será registrado.

O bloco `datasourcesIntegration.openmetadata` define como o job autentica na API do OpenMetadata para realizar o registro.
Quando `datasourcesIntegration.openmetadata.auth.passwordSecret.enabled: true`, informe o nome do Secret e a chave da senha no arquivo de valores.

> **CREDENCIAIS PADRÃO DE SUPERSET E AIRFLOW**
>
> Superset e Airflow usam `admin`/`admin` como credenciais padrão no arquivo de valores de exemplo. Altere em produção via `connection.username`/`connection.password` nos values ou, preferencialmente, via Secret Kubernetes.

Arquivos de valores modulares

O chart disponibiliza arquivos de valores separados por serviço, que podem ser usados individualmente ou combinados em um mesmo comando Helm com múltiplas opções -f.

© Copyright Tecnisys 2026. All rights reserved.

Arquivo	Integração habilitada
`values-datasources-integration.yaml`	Todas as integrações de uma vez
`values-trino-integration.yaml`	Somente Trino
`values-clickhouse-integration.yaml`	Somente ClickHouse
`values-superset-integration.yaml`	Somente Superset
`values-airflow-integration.yaml`	Somente Airflow
`values-kafka-integration.yaml`	Somente Kafka

Terminal input

```
# Somente Trino
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> -f values-trino-integration.yaml

# Trino + ClickHouse
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> -f values-trino-integration.yaml -f values-clickhouse-
integration.yaml

# Todas as integrações de uma vez
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> -f values-datasources-integration.yaml
```

Configuração por serviço

Cada integração possui um campo `enabled`, campos `serviceName`/`displayName` (nome e rótulo visível na UI) e um bloco `connection` sob `datasourcesIntegration.*`.

Trino e ClickHouse

© Copyright Tecnisys 2026. All rights reserved.

```
datasourcesIntegration:
  enabled: true
  openmetadata:
    apiEndpoint: "http://<OPENMETADATA_SERVICE>.
<NAMESPACE>.svc.cluster.local:8585/api"
    auth:
      provider: "basic"
      username: "admin@open-metadata.org"
      passwordSecret:
        enabled: true
        name: "<SECRET_NAME>"
        key: "<SECRET_KEY>"

  trino:
    enabled: true
    serviceName: "tdp-trino"
    displayName: "TDP Trino"
    connection:
      host: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 8080
      username: "<USERNAME>"
      scheme: "http"

  clickhouse:
    enabled: true
    serviceName: "tdp-clickhouse"
    displayName: "TDP ClickHouse"
    connection:
      host: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 8123
      username: "<USERNAME>"
      passwordSecret:
        name: "<SECRET_NAME>"
        key: "password"
      scheme: "http"
```

O Secret do ClickHouse deve existir antes do deploy: o nome padrão é `tdp-clickhouse-tdp-user`, chave `password`.

> **AMBIENTE DE PRODUÇÃO (TLS)**

© Copyright Tecnisys 2026. All rights reserved.



> Para ambientes com TLS, use `scheme: "https"` e a porta segura (ex.: `8443` para ClickHouse). Para Trino com TLS, defina `scheme: "https"` e ajuste a porta. Consulte Segurança — OpenMetadata para exemplos completos.

Superset

O Superset é registrado como *Dashboard Service*. O parâmetro `hostPort` inclui o protocolo e a porta.

```
datasourcesIntegration:
  superset:
    enabled: true
    serviceName: "tdp-superset"
    displayName: "TDP Superset"
    connection:
      hostPort: "http://tdp-superset.<NAMESPACE>.svc.cluster.local:8088"
      username: "<USERNAME>"
      password: "<PASSWORD>"
      provider: "db"
      verifySSL: "no-ssl"
```

> **NOTE**
>
> Use `provider: "ldap"` se o Superset estiver configurado com LDAP. Para HTTPS, altere `verifySSL` para `"validate"` e configure o certificado.

Airflow

O Airflow é registrado como *Pipeline Service*. Use o endpoint da API do servidor Airflow.

```
datasourcesIntegration:
  airflow:
    enabled: true
    serviceName: "tdp-airflow"
    displayName: "TDP Airflow"
    connection:
      hostPort: "http://tdp-airflow-api-server.
<NAMESPACE>.svc.cluster.local:8080"
      numberOfStatus: 10
```

© Copyright Tecnisys 2026. All rights reserved.

```
      username: "<USERNAME>"
      password: "<PASSWORD>"
      verifySSL: "no-ssl"
```

Kafka

O Kafka é registrado como *Messaging Service*. Informe o endereço do bootstrap server; para múltiplos brokers, separe por vírgula.

```
datasourcesIntegration:
  kafka:
    enabled: true
    serviceName: "tdp-kafka"
    displayName: "TDP Kafka"
    connection:
      bootstrapServers: "<KAFKA_BOOTSTRAP_SERVICE>.
<NAMESPACE>.svc.cluster.local:9092"
```

Para Kafka com autenticação SASL, configure também `saslUsername`, `saslPassword`, `saslMechanism` e `securityProtocol` no bloco `connection`.

Execução de pipelines: Airflow ou K8S

O OpenMetadata pode executar pipelines de ingestão usando dois modos:

- **Kubernetes Jobs** (`k8s`): modo padrão, no qual os pipelines rodam como Jobs nativos no cluster, sem dependência de Airflow
- **Airflow** (`airflow`): modo em que os pipelines são executados por integração REST com o serviço Airflow

O modo de execução é controlado pelo campo `pipelineServiceClientConfig.type`, que aceita os valores `k8s` ou `airflow`.

Modo Kubernetes Jobs (`type: k8s`)

Quando o modo `k8s` está ativo, os pipelines são submetidos como Jobs Kubernetes. Os parâmetros mais comuns nesse cenário são:

- imagem utilizada na execução
- namespace onde os Jobs serão criados

© Copyright Tecnisys 2026. All rights reserved.



- service account
- recursos de CPU e memória
- TTL e política de *backoff*
- `nodeSelector`, `annotations` e `securityContext`

Modo Airflow (`type: airflow`)

Quando o modo `airflow` está ativo, o OpenMetadata delega a execução dos pipelines ao serviço Airflow. Os parâmetros mais comuns nesse cenário são:

- endpoint do Airflow
- usuário
- senha via Secret
- verificação SSL

Como alternar

- altere apenas o valor de `pipelineServiceClientConfig.type` para trocar o modo de execução
- mantenha os parâmetros específicos alinhados ao modo escolhido
- o modo `airflow` exige que o serviço Airflow esteja habilitado e acessível nas dependências do chart

Verificação

Após o Job concluir:

1. Acesse a UI do OpenMetadata (porta-forward ou Ingress).
2. Navegue até **Settings** → **Services**.
3. Verifique os serviços registrados por tipo:
   - **Databases**: TDP ClickHouse, TDP Trino
   - **Dashboards**: TDP Superset
   - **Pipelines**: TDP Airflow
   - **Messaging**: TDP Kafka
4. Para testar a conexão: abra o serviço, clique em **Test Connection** e confirme o sucesso.
5. Para iniciar ingestão de metadados: abra o serviço, clique em **Add Ingestion** e configure o workflow.

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> get jobs | grep register-datasources
kubectl -n <NAMESPACE> logs job/<RELEASE_NAME>-register-datasources
```

Re-executar o Job

O Job é um hook `post-install`/`post-upgrade`. Para re-executá-lo:

Terminal input

```
# Remover o Job existente
kubectl delete job -n <NAMESPACE> <RELEASE_NAME>-register-datasources

# Re-executar via helm upgrade (o hook recria o Job)
helm upgrade <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata \
  -n <NAMESPACE> \
  -f <VALUES_FILE>
```

Habilitar e desabilitar serviços individualmente

Para desabilitar um serviço sem remover sua configuração:

```
datasourcesIntegration:
  clickhouse:
    enabled: false
  trino:
    enabled: true
  superset:
    enabled: true
```

Para desabilitar o Job inteiro (nenhum serviço é registrado):

```
datasourcesIntegration:
  enabled: false
```

Configuração avançada do Job

© Copyright Tecnisys 2026. All rights reserved.

Retentativas e TTL

```
datasourcesIntegration:
  job:
    backoffLimit: 5
    ttlSecondsAfterFinished: 600
```

Imagem do Job

```
datasourcesIntegration:
job:
  image:
    repository: "docker.getcollate.io/openmetadata/ingestion"
    tag: "1.12.4"
    pullPolicy: "IfNotPresent"
```

Recursos do Job

```
datasourcesIntegration:
  job:
    resources:
      requests:
        cpu: 200m
        memory: 512Mi
      limits:
        cpu: 1
        memory: 1Gi
```

Autenticação JWT

Se o OpenMetadata estiver configurado com JWT em vez de autenticação básica, altere o provider no bloco de autenticação do job:

```
datasourcesIntegration:
  openmetadata:
    auth:
      provider: "jwt"
      # O Job deve ser ajustado para utilizar JWT tokens no lugar de
usuário/senha
```

© Copyright Tecnisys 2026. All rights reserved.

Exemplos de configuração

Ambiente de desenvolvimento

Configuração mínima para um ambiente local ou de desenvolvimento, com ClickHouse e Trino acessíveis no cluster, sem TLS e com credenciais simples no arquivo de valores:

```
datasourcesIntegration:
  enabled: true

  openmetadata:
    apiEndpoint: "http://<RELEASE_NAME>.<NAMESPACE>.svc.cluster.local:8585/api"
    auth:
      provider: "basic"
      username: "admin@open-metadata.org"
      password: "admin"

  clickhouse:
    enabled: true
    serviceName: "tdp-clickhouse"
    displayName: "TDP ClickHouse"
    connection:
      host: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 8123
      scheme: "http"
      username: "tdpuser"

  trino:
    enabled: true
    serviceName: "tdp-trino"
    displayName: "TDP Trino"
    connection:
      host: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 8080
      scheme: "http"
      username: "admin"
```

Ambiente de produção

Configuração para produção com TLS ativado, credenciais via Secrets Kubernetes e endpoint HTTPS do OpenMetadata:

© Copyright Tecnisys 2026. All rights reserved.

```
datasourcesIntegration:
  enabled: true

  openmetadata:
    apiEndpoint: "https://<OPENMETADATA_HOSTNAME>/api"
    auth:
      provider: "basic"
      username: "admin@open-metadata.org"
      passwordSecret:
        enabled: true
        name: "<SECRET_NAME>"
        key: "password"

  clickhouse:
    enabled: true
    serviceName: "tdp-clickhouse"
    displayName: "TDP ClickHouse"
    connection:
      host: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 8443
      scheme: "https"
      username: "tdpuser"
      passwordSecret:
        name: "tdp-clickhouse-tdp-user"
        key: "password"

  trino:
    enabled: true
    serviceName: "tdp-trino"
    displayName: "TDP Trino"
    connection:
      host: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: <SERVICE_PORT>
      scheme: "https"
      username: "admin"

  superset:
    enabled: true
    serviceName: "tdp-superset"
    displayName: "TDP Superset"
    connection:
      hostPort: "https://tdp-superset.<NAMESPACE>.svc.cluster.local:8088"
      username: "admin"
      passwordSecret:
```

© Copyright Tecnisys 2026. All rights reserved.

```
      name: "<SECRET_NAME>"
      key: "password"
    provider: "db"
    verifySSL: "validate"
```

> **NOTE**
>
> Em produção, nunca armazene senhas diretamente no arquivo de valores. Use sempre `passwordSecret` com referência a um Kubernetes Secret criado previamente.

Solução de problemas

Terminal input

```
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<RELEASE_NAME>-register-datasources
kubectl -n <NAMESPACE> describe job <RELEASE_NAME>-register-datasources
```

> **API REST DO OPENMETADATA**
>
> O job usa a API REST do OpenMetadata para registrar serviços. Endpoints principais:
>
> - `POST /api/v1/services/databaseServices` — criar serviço
> - `PUT /api/v1/services/databaseServices` — atualizar serviço
> - `GET /api/v1/services/databaseServices/name/{name}` — consultar por nome
>
> Documentação completa: docs.open-metadata.org/swagger.html

Sintoma	Causa provável	Correção
Job falha: "OpenMetadata not ready"	OpenMetadata ainda inicializando; o Job aguarda	Verifique pods: `kubectl get pods -n <NAMESPACE>`

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Causa provável	Correção
	até 30 tentativas (aprox. 5 min)	
Job falha: "Authentication error"	Credenciais incorretas no arquivo de valores	Confirme `username`/`password` em `datasourcesIntegration.openmetadata.auth`
Job falha: "Secret not found"	Secret do ClickHouse ausente	Confirme: `kubectl get secret -n <NAMESPACE> tdp-clickhouse-tdp-user`
Job falha: "Service unreachable"	Hostname ou porta incorretos	Verifique `host`/`hostPort`/`bootstrapServers` nos values
Serviço não aparece na UI	Job não concluiu ou falhou silenciosamente	Verifique logs e status do Job

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone

CHART

Version 3.0.1 | Type application | AppVersion 2.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado | Ingress suportado | Gateway API suportado

O que é o Apache Ozone no TDP?

O Apache Ozone é o armazenamento de objetos compatível com S3 usado pelo TDP Kubernetes para disponibilizar um backend de dados dentro do cluster. Ele atende workloads que gravam ou leem objetos, como jobs Spark, catálogos Trino, tabelas Iceberg ou Delta Lake e clientes compatíveis com S3.

Estrutura de valores (Helm)

Os valores do Apache Ozone ficam sob a chave `tdp-ozone:` (alias da dependência no `Chart.yaml`); as configurações internas do subchart ficam aninhadas sob `ozone:`. Configurações de exposição HTTP/HTTPS ficam sob `TDP-Settings:` e `ingress:`.

```
TDP-Settings:
  gateway:
    ingress:
      enabled: false

ozone:
  datanode:
    replicas: 3
    persistence:
      enabled: true
      size: 50Gi
```

© Copyright Tecnisys 2026. All rights reserved.

Arquitetura

O deploy do Ozone é composto por serviços com papéis distintos:

Serviço	Papel	Porta padrão
S3 Gateway REST	Plano de dados compatível com API S3 para leitura e escrita de objetos	`9878`
S3 Gateway Web UI	Interface web do S3 Gateway para inspeção operacional	`19878`
Ozone Manager (OM)	Gerencia volumes, buckets, chaves e metadados de namespace	`9874`
Storage Container Manager (SCM)	Coordena containers, pipelines e localização dos blocos nos datanodes	`9876`
Datanodes	Armazenam os dados de objeto em volumes persistentes ou efêmeros	`9882`

As aplicações normalmente usam apenas o S3 Gateway REST. OM, SCM e S3 Gateway Web UI são interfaces administrativas ou de observabilidade e devem ser expostos com mais restrição.

Visão geral

Item	Valor
Chart	`tdp-ozone`
Versão da aplicação	Apache Ozone 2.0.0

© Copyright Tecnisys 2026. All rights reserved.

Item	Valor
Registry OCI	`oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone`
Endpoint interno S3	`http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878`

Páginas relacionadas

- Integrações - Apache Ozone explica como consumidores TDP usam o endpoint S3.
- Segurança - Apache Ozone detalha AWS Signature v4, Secrets, TLS e rotação de credenciais.
- Ingress - Apache Ozone mostra a exposição de OM, SCM, S3 Web UI e S3 REST.

Pré-requisitos

Antes de instalar, confirme:

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível para os componentes que usarão persistência.
- Ingress Controller ou Gateway API apenas quando houver exposição HTTP/HTTPS fora do cluster.
- Secret de credenciais S3 quando `ozone.s3g.auth.enabled` permanecer habilitado.

Instalação

Instalação

Durante a instalação, o chart cria automaticamente o Secret padrão `ozone-s3-credentials` com as credenciais S3. A criação manual só é necessária quando você quiser usar credenciais personalizadas — nesse caso, crie o Secret antes do deploy:

Terminal input

```
kubectl -n <NAMESPACE> create secret generic ozone-s3-credentials --from-literal=aws_access_key_id="<AWS_ACCESS_KEY_ID>" --from-
```

© Copyright Tecnisys 2026. All rights reserved.

```
literal=aws_secret_access_key="<AWS_SECRET_ACCESS_KEY>"
```

Instale ou atualize o chart com um arquivo de valores versionado para o seu ambiente:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone -n <NAMESPACE> --create-
namespace -f <VALUES_FILE>
```

OpenShift

No OpenShift, o chart requer permissões adicionais. Habilite o suporte à plataforma ao instalar:

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone -n <NAMESPACE> --create-
namespace --set openshift.enabled=true -f <VALUES_FILE>
```

Para validar a criação dos recursos principais:

Terminal input

```
kubectl -n <NAMESPACE> get pods,svc,ingress
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão
`ozone.enabled`	Habilita o subchart do Ozone	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`ozone.image.repository`	Imagem usada pelos serviços Ozone	`apache/ozone`
`ozone.image.tag`	Versão da imagem Ozone	`2.0.0`
`ozone.datanode.replicas`	Quantidade de datanodes	`3`
`ozone.datanode.persistence.enabled`	Habilita persistência dos Datanodes	`false`
`ozone.om.replicas`	Réplicas do Ozone Manager	`1`
`ozone.om.service.type`	Tipo de Service do Ozone Manager	`ClusterIP`
`ozone.scm.replicas`	Réplicas do SCM	`1`
`ozone.scm.service.type`	Tipo de Service do SCM	`ClusterIP`
`ozone.s3g.replicas`	Réplicas do S3 Gateway	`1`
`ozone.s3g.auth.enabled`	Habilita autenticação AWS Signature v4 no S3 Gateway	`true`
`ozone.s3g.auth.secretName`	Secret Kubernetes com `aws_access_key_id` e `aws_secret_access_key`	`ozone-s3-credentials`
`ingress.om.enabled`	Expõe a UI do Ozone Manager por Ingress	`true`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`ingress.om.hosts[0].host`	Hostname do Ingress para UI do OM	`ozone-om.tdp.local`
`ingress.scm.enabled`	Expõe a UI do SCM por Ingress	`true`
`ingress.scm.hosts[0].host`	Hostname do Ingress para UI do SCM	`ozone-scm.tdp.local`
`ingress.s3g.web.enabled`	Expõe a Web UI do S3 Gateway por Ingress	`true`
`ingress.s3g.web.hosts[0].host`	Hostname do Ingress para Web UI do S3 Gateway	`ozone-s3-ui.tdp.local`
`ingress.s3g.rest.enabled`	Expõe a API REST S3 por Ingress	`true`
`ingress.s3g.rest.hosts[0].host`	Hostname do Ingress para API REST S3	`ozone-s3.tdp.local`
`TDP-Settings.gateway.ingress.enabled`	Habilita recursos Ingress do chart	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita recursos Gateway API do chart	`false`

Detalhes de configuração

Persistência

Por padrão, todos os serviços iniciam sem volumes persistentes (`persistence.enabled: false`). Em produção, habilite PVCs ao menos nos componentes que mantêm estado:

© Copyright Tecnisys 2026. All rights reserved.

Datanodes, OM e SCM.

Para cada componente, o padrão de configuração é o mesmo — basta habilitar e informar o tamanho desejado. O campo `storageClassName` é opcional: se omitido, o cluster usa a StorageClass marcada como padrão.

```
ozone:
  datanode:
    persistence:
      enabled: true
      size: 100Gi
      # storageClassName: "<STORAGE_CLASS>"  # opcional
  om:
    persistence:
      enabled: true
      size: 10Gi
  scm:
    persistence:
      enabled: true
      size: 10Gi
```

Dimensione os datanodes conforme o crescimento esperado do lakehouse. OM e SCM precisam de bem menos espaço, mas devem ter disco estável para preservar metadados e histórico operacional.

Acesso & Segurança

Acesso

S3 Gateway REST

Dentro do cluster, use o Service do S3 Gateway REST:

```
http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878
```

Para testes locais sem Ingress, use port-forward:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME>-s3g-rest 9878:9878
```

Depois aponte clientes S3 para `http://127.0.0.1:9878`.

Interfaces administrativas

Quando o Ingress estiver habilitado, as interfaces web são acessíveis pelos hostnames configurados em `ingress.*.hosts[0].host`. Os valores padrão do chart são:

Interface	Hostname padrão
S3 Gateway Web UI	`ozone-s3-ui.tdp.local`
Ozone Manager UI	`ozone-om.tdp.local`
SCM UI	`ozone-scm.tdp.local`

Sem Ingress, use port-forward. Os nomes de service são baseados no nome do release (`<RELEASE_NAME>`):

Interface	Service	Porta
S3 Gateway Web UI	`<RELEASE_NAME>-s3g-web`	`19878`
Ozone Manager UI	`<RELEASE_NAME>-om`	`9874`
SCM UI	`<RELEASE_NAME>-scm`	`9876`

> **WARNING**
>
> Ao expor o Ozone fora do cluster, trate o S3 Gateway REST como API de dados: exija TLS, autenticação S3, DNS correto e NetworkPolicy. Exponha apenas quando o acesso externo for estritamente necessário.

Solução de problemas

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Sintoma	Verificação	Ação
Pods não ficam prontos	`kubectl -n <NAMESPACE> get pods`	Verifique eventos, recursos e PVCs pendentes
Services ausentes	`kubectl -n <NAMESPACE> get svc`	Confirme o nome do release e se o chart foi instalado no namespace correto
Cliente S3 não conecta	`kubectl -n <NAMESPACE> get endpoints <RELEASE_NAME>-s3g-rest`	Valide DNS interno, porta `9878` e NetworkPolicy
Erro de autenticação S3	`kubectl -n <NAMESPACE> get secret ozone-s3-credentials`	Confira o Secret e as credenciais usadas pelo cliente
Ingress não responde	`kubectl -n <NAMESPACE> get ingress`	Revise `ingressClassName`, DNS e TLS do hostname publicado

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 2.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

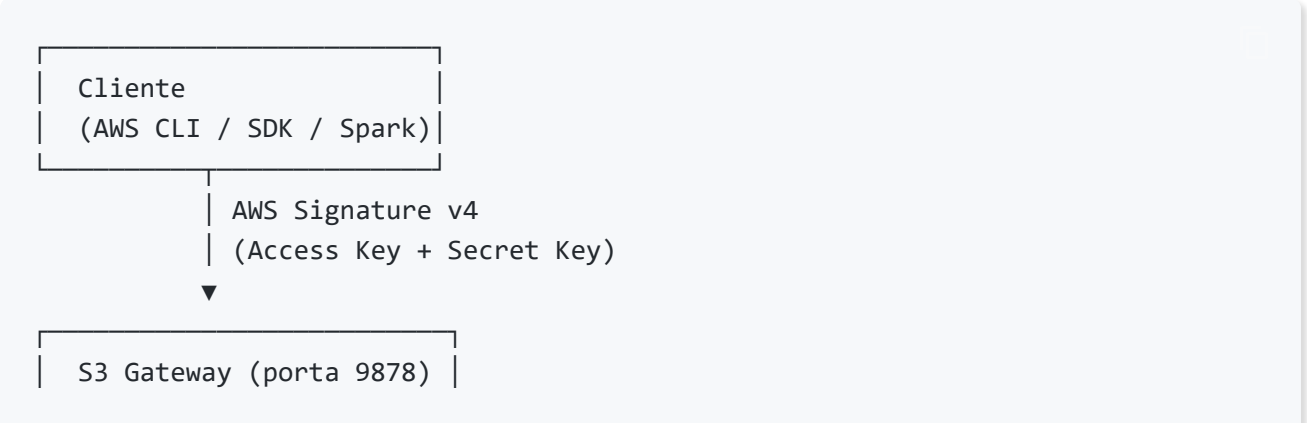
> **PADRÕES GERAIS DE SEGURANÇA**
>
> Para conceitos transversais a todos os componentes TDP — como gerenciamento de Secrets, rotação de credenciais, NetworkPolicy, rate limiting e auditoria — consulte Segurança — Padrões Gerais.

Visão geral

O S3 Gateway do Apache Ozone no TDP usa autenticação AWS Signature Version 4 em modo simples, sem Kerberos. Nesse modo, os clientes assinam cada requisição com um access key e um secret key, e o chart injeta as credenciais no S3 Gateway por meio de Secret Kubernetes.

A segurança efetiva combina quatro controles: credenciais por aplicação ou usuário técnico, TLS quando houver tráfego fora do cluster, restrição de rede com NetworkPolicy ou controles equivalentes e auditoria operacional por logs.

Arquitetura de segurança



© Copyright Tecnisys 2026. All rights reserved.

```
│  - Valida assinatura      │
│  - Verifica credenciais   │
└─────────────┬─────────────┘
              │
              ▼
┌─────────────────────────────┐
│  Ozone Manager              │
│  (Autorização)              │
└─────────────────────────────┘
```

AWS Signature v4 sem Kerberos

Níveis de segurança disponíveis

Nível	Descrição	Uso recomendado
None	Sem autenticação	Desenvolvimento isolado apenas
Simple	AWS Signature v4 sem Kerberos	**Produção (recomendado)**
Kerberos	Segurança Ozone completa	Ambientes enterprise com Kerberos existente

Configuração no values.yaml

```yaml
ozone:
  s3g:
    auth:
      enabled: true
      type: simple
      secretName: ozone-s3-credentials
    env:
      - name: OZONE-SITE.XML_ozone.s3g.authentication
        value: "simple"
      - name: OZONE-SITE.XML_ozone.security.enabled
        value: "false"
```

© Copyright Tecnisys 2026. All rights reserved.

```
      - name: OZONE-SITE.XML_ozone.s3g.secret.http.enabled
        value: "false"
```

Parâmetro	Descrição	Padrão
`ozone.s3g.auth.enabled`	Habilita autenticação S3 no S3 Gateway	`true`
`ozone.s3g.auth.secretName`	Secret com as chaves `aws_access_key_id` e `aws_secret_access_key`	`ozone-s3-credentials`

Com autenticação habilitada, clientes como AWS CLI, SDKs, Spark S3A e Trino precisam usar a mesma região, endpoint e credenciais esperadas pelo S3 Gateway. Use `us-east-1` como região padrão quando não houver outra convenção no ambiente.

> **DESENVOLVIMENTO APENAS**
>
> Use `ozone.s3g.auth.enabled: false` somente em ambientes de desenvolvimento ou laboratório isolado. Em ambientes compartilhados ou produtivos, mantenha autenticação habilitada e proteja o endpoint S3 REST com TLS e controle de rede.

Modelo de credenciais

Crie credenciais separadas por aplicação, pipeline ou usuário técnico sempre que possível. Evite compartilhar a mesma credencial entre consumidores com perfis de acesso diferentes.

Terminal input

```
kubectl -n <NAMESPACE> create secret generic ozone-s3-credentials \
  --from-literal=aws_access_key_id="<AWS_ACCESS_KEY_ID>" \
  --from-literal=aws_secret_access_key="<AWS_SECRET_ACCESS_KEY>"
```

© Copyright Tecnisys 2026. All rights reserved.



O Secret deve existir no namespace em que o Ozone é instalado e deve ser referenciado pelo valor `ozone.s3g.auth.secretName`:

```
ozone:
  s3g:
    auth:
      enabled: true
      secretName: ozone-s3-credentials
```

Não versionar credenciais reais em repositórios Git, arquivos de valores, notebooks ou DAGs. Distribua credenciais para os consumidores por Secrets Kubernetes, variáveis de ambiente gerenciadas ou mecanismos equivalentes do componente.

Gerenciamento de usuários S3

Quando o ambiente usa credenciais separadas por pessoa, aplicação ou serviço técnico, padronize um fluxo operacional para criar, listar e remover usuários S3.

Criação via script

O script `scripts/generate-s3-credentials.sh` fica dentro do chart `tdp-ozone`. Para usá-lo, execute o comando a partir da raiz do chart, ou ajuste o caminho para o local onde o chart foi extraído. O utilitário gera pares de access key e secret key e os registra como Secrets Kubernetes:

Terminal input

```
NAMESPACE=<NAMESPACE> ./scripts/generate-s3-credentials.sh create
<APPLICATION_NAME>
```

Saída esperada:

```
AWS_ACCESS_KEY_ID=a1b2c3d4e5f6g7h8i9j0
AWS_SECRET_ACCESS_KEY=aBcDeFgHiJkLmNoPqRsTuVwXyZ1234567890ABCD
```

Criação manual

Se preferir criar as credenciais manualmente sem o script:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
ACCESS_KEY=$(openssl rand -hex 10)
SECRET_KEY=$(openssl rand -base64 32)

kubectl create secret generic ozone-s3-credentials-<APPLICATION_NAME> \
  --from-literal=aws_access_key_id="$ACCESS_KEY" \
  --from-literal=aws_secret_access_key="$SECRET_KEY" \
  -n <NAMESPACE>
```

Após criar o Secret, registre o `accessId` no Ozone Manager para associá-lo a um usuário do tenant — sem este passo, as credenciais existem no Kubernetes mas não são reconhecidas pelo S3 Gateway:

Terminal input

```
kubectl exec -n <NAMESPACE> <RELEASE_NAME>-om-0 -- \
  ozone sh tenant assignusertoaccessid \
  --accessid="$ACCESS_KEY" --tenant-id=default --user-principal=
<APPLICATION_NAME>
```

Listagem

Terminal input

```
./scripts/generate-s3-credentials.sh list
```

Para listar os Secrets do S3 Gateway diretamente no cluster:

Terminal input

```
kubectl get secrets -n <NAMESPACE> -l app.kubernetes.io/component=s3g
```

Remoção

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
./scripts/generate-s3-credentials.sh delete <APPLICATION_NAME>
```

Para auditoria operacional, mantenha registro de qual Secret Kubernetes distribui cada credencial e quais workloads consomem esse Secret.

Controle de acesso (ACLs)

O Ozone oferece ACLs em nível de volume e bucket. Use o CLI `ozone sh` para definir e consultar permissões:

Terminal input

```
# Definir ACL de bucket (alice com leitura/escrita, bob somente leitura)
kubectl exec -n <NAMESPACE> <RELEASE_NAME>-om-0 -- \
  ozone sh bucket setacl \
  --acl user:alice:rw,user:bob:r \
  /volume1/bucket1
```

Terminal input

```
# Consultar ACLs de um bucket
kubectl exec -n <NAMESPACE> <RELEASE_NAME>-om-0 -- \
  ozone sh bucket getacl /volume1/bucket1
```

Rotação e revogação

Para rotacionar credenciais, atualize ou recrie o Secret e reinicie os pods que consomem essas variáveis ou arquivos. Isso inclui o S3 Gateway e os clientes que mantêm credenciais em memória.

Terminal input

```
kubectl -n <NAMESPACE> rollout restart statefulset/<RELEASE_NAME>-s3g
```

A revogação prática em modo simples é operacional: remover ou trocar Secrets dos namespaces consumidores, reiniciar workloads e bloquear caminhos de rede quando

© Copyright Tecnisys 2026. All rights reserved.

necessário.

LDAP/AD sem Kerberos

LDAP ou Active Directory podem ser usados como referência de ciclo de vida e nomenclatura de contas, por exemplo para padronizar usuários técnicos.

> **LIMITAÇÃO IMPORTANTE**
>
> Com `ozone.security.enabled=false` (sem Kerberos), desabilitar uma conta no LDAP/AD **não revoga automaticamente** credenciais S3 já distribuídas. As chaves S3 são credenciais de cliente que você distribui — a revogação é responsabilidade operacional.

Processo recomendado para gestão de ciclo de vida com LDAP/AD:

1. Mantenha um identificador estável por conta (ex.: `uid` do LDAP ou `sAMAccountName` do AD).
2. Para cada conta LDAP, gere um par de chaves S3 e armazene como Kubernetes Secret.
3. Injete o Secret nos workloads que precisam de acesso (env vars ou arquivos montados).
4. Quando uma conta for desativada no LDAP, remova o Secret Kubernetes correspondente de todos os namespaces e reimplante os workloads.
5. Rotacione chaves periodicamente ou imediatamente após comprometimento.

Controles compensatórios recomendados:

- Restrinja o acesso ao endpoint S3 com **NetworkPolicy** ou security groups
- Use **TLS** para o tráfego S3
- Crie service accounts separados (chaves distintas) por aplicação
- Habilite audit logging no Ozone

TLS e exposição do S3 REST

Use TLS para qualquer acesso ao S3 Gateway REST fora do cluster ou em redes compartilhadas. O TLS normalmente é terminado no Ingress Controller ou no Gateway configurado para o hostname `https://<OZONE_S3_REST_HOSTNAME>`.

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-ozone:
  ingress:
    s3g:
      rest:
        enabled: true
        annotations:
          cert-manager.io/cluster-issuer: letsencrypt-prod
        hosts:
          - host: <OZONE_S3_REST_HOSTNAME>
            paths:
              - path: /
                pathType: Prefix
        tls:
          - secretName: ozone-s3-rest-tls
            hosts:
              - <OZONE_S3_REST_HOSTNAME>
```

Exponha OM, SCM e S3 Web UI apenas para redes administrativas. O S3 Gateway REST é plano de dados e exige atenção adicional a TLS, autenticação S3, DNS e políticas de rede.

Rate limiting

Para proteger o endpoint S3 REST contra uso abusivo, configure limites de requisição no Ingress Controller NGINX:

```
tdp-ozone:
  ingress:
    s3g:
      rest:
        annotations:
          nginx.ingress.kubernetes.io/limit-rps: "100"
          nginx.ingress.kubernetes.io/limit-connections: "10"
```

NetworkPolicy e segurança de rede

Restrinja quem pode alcançar o Service `<RELEASE_NAME>-s3g-rest` na porta `9878`. Em clusters com NetworkPolicy, permita somente namespaces e ServiceAccounts dos consumidores autorizados, como Spark, Trino, Hive Metastore, Airflow, NiFi ou Jupyter quando aplicável.

© Copyright Tecnisys 2026. All rights reserved.

Também revise exposição externa: se todos os consumidores estão dentro do cluster, prefira Service interno em vez de Ingress para o S3 REST.

Auditoria e logs

Habilite o audit logging no `values.yaml`:

```
ozone:
  s3g:
    env:
      - name: OZONE-SITE.XML_ozone.audit.enabled
        value: "true"
      - name: OZONE-SITE.XML_ozone.audit.log.file
        value: "/var/log/ozone/s3-audit.log"
```

Para consultar entradas de auditoria nos logs do S3 Gateway:

Terminal input

```
kubectl logs -n <NAMESPACE> statefulset/<RELEASE_NAME>-s3g | grep AUDIT
```

Use logs do S3 Gateway e dos consumidores para investigar acessos, falhas de assinatura e erros de autorização. Quando o audit logging estiver habilitado, direcione esses logs para a solução de observabilidade adotada pelo TDP.

Checklist de produção

Antes de expor o S3 Gateway REST para usuários ou integrações fora do cluster, valide:

- [ ] Credenciais padrão substituídas por credenciais próprias do ambiente
- [ ] TLS habilitado para o hostname público do S3 REST
- [ ] Anotação cert-manager ou certificado provisionado manualmente
- [ ] Política de rotação definida para credenciais S3
- [ ] Procedimento de revogação e recuperação de credenciais documentado
- [ ] NetworkPolicy, regras de firewall ou controles equivalentes restringindo o acesso ao endpoint
- [ ] Rate limiting configurado no Ingress (se exposto externamente)
- [ ] Audit logging habilitado e logs direcionados para observabilidade

© Copyright Tecnisys 2026. All rights reserved.

- [ ] Teste de criação, leitura, gravação e remoção de objetos com as credenciais finais

Migração de acesso aberto para autenticado

Se o ambiente está migrando de uma instalação sem autenticação (`auth.enabled: false`) para autenticação habilitada:

1. **Crie as credenciais antes de habilitar a autenticação:**

   Terminal input

   ```
   NAMESPACE=<NAMESPACE> ./scripts/generate-s3-credentials.sh create
   <APPLICATION_NAME>
   ```

2. **Atualize o `values.yaml`** habilitando `ozone.s3g.auth.enabled: true` e referenciando o Secret criado.

3. **Aplique a atualização do chart:**

   Terminal input

   ```
   helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
   ozone \
     --namespace <NAMESPACE> \
     -f values.yaml
   ```

4. **Atualize todos os consumidores** (Spark, Trino, Airflow, etc.) com as novas credenciais antes ou imediatamente após o upgrade.

5. **Verifique** que todas as aplicações estão operando com as novas credenciais antes de remover o acesso aberto.

Teste de acesso

Verifique a autenticação com e sem credenciais:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
# Sem credenciais — deve retornar erro de autenticação
curl http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878/
```

Terminal input

```
# Com credenciais — deve retornar lista de buckets
aws configure set aws_access_key_id <AWS_ACCESS_KEY_ID>
aws configure set aws_secret_access_key <AWS_SECRET_ACCESS_KEY>
aws configure set region us-east-1
aws s3 ls --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
```

Solução de problemas

`SignatureDoesNotMatch`

1. Verifique que as credenciais no Secret estão corretas:

   Terminal input

   ```
   kubectl get secret ozone-s3-credentials-<APPLICATION_NAME> -n <NAMESPACE> -o
   jsonpath='{.data}'
   ```

2. Confirme que o relógio do nó está sincronizado (AWS Signature v4 é sensível a desvio de tempo).

3. Verifique que o endpoint usado pelo cliente é exatamente o mesmo usado na assinatura.

4. Confirme que a região está configurada como `us-east-1`.

Credenciais não reconhecidas (`AccessDenied` ou 403)

Terminal input

```
./scripts/generate-s3-credentials.sh create <APPLICATION_NAME>
```

© Copyright Tecnisys 2026. All rights reserved.

Reinicie os pods consumidores após recriar o Secret.

Conexão recusada

1. Verifique se o Ingress está configurado corretamente.

2. Teste resolução DNS interna:

Terminal input

```
kubectl exec -it <POD_NAME> -n <NAMESPACE> -- nslookup <RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local
```

3. Teste via port-forward:

Terminal input

```
kubectl port-forward -n <NAMESPACE> svc/<RELEASE_NAME>-s3g-rest 9878:9878
aws s3 ls --endpoint-url=http://localhost:9878
```

`AccessDenied` com credencial válida

Verifique ACLs do volume ou bucket:

Terminal input

```
kubectl exec -n <NAMESPACE> <RELEASE_NAME>-om-0 -- \
  ozone sh bucket getacl /<VOLUME_NAME>/<S3_BUCKET>
```

Ajuste permissões conforme necessário com `ozone sh bucket setacl`.

Sintoma	Causa provável	Correção
`SignatureDoesNotMatch`	Credencial incorreta, endpoint divergente, região errada ou clock drift	Confira Secret, URL exata, região `us-east-1` e sincronização de horário

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Causa provável	Correção
`AccessDenied` ou 403	Credenciais não reconhecidas ou política do cliente inconsistente	Valide o Secret e reinicie pods consumidores
`AccessDenied` com credencial válida	ACL de volume ou bucket não permite a operação	Verifique ACLs com `ozone sh bucket getacl`
Cliente conecta por HTTP mas falha por HTTPS	Certificado, CA ou truststore ausente no consumidor	Configure CA/truststore e use o mesmo esquema da URL publicada
Funciona via port-forward, falha via Ingress	DNS, TLS, IngressClass ou regra de rota incorreta	Verifique hostname, certificado, Ingress e logs do controller
Consumidor interno não alcança o endpoint	NetworkPolicy ou namespace bloqueando tráfego	Libere o tráfego necessário para `<RELEASE_NAME>-s3g-rest:9878`

Referências

- Apache Ozone S3 Gateway — Documentação oficial
- AWS Signature Version 4
- Ozone Security Architecture

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone – Integração

CHART

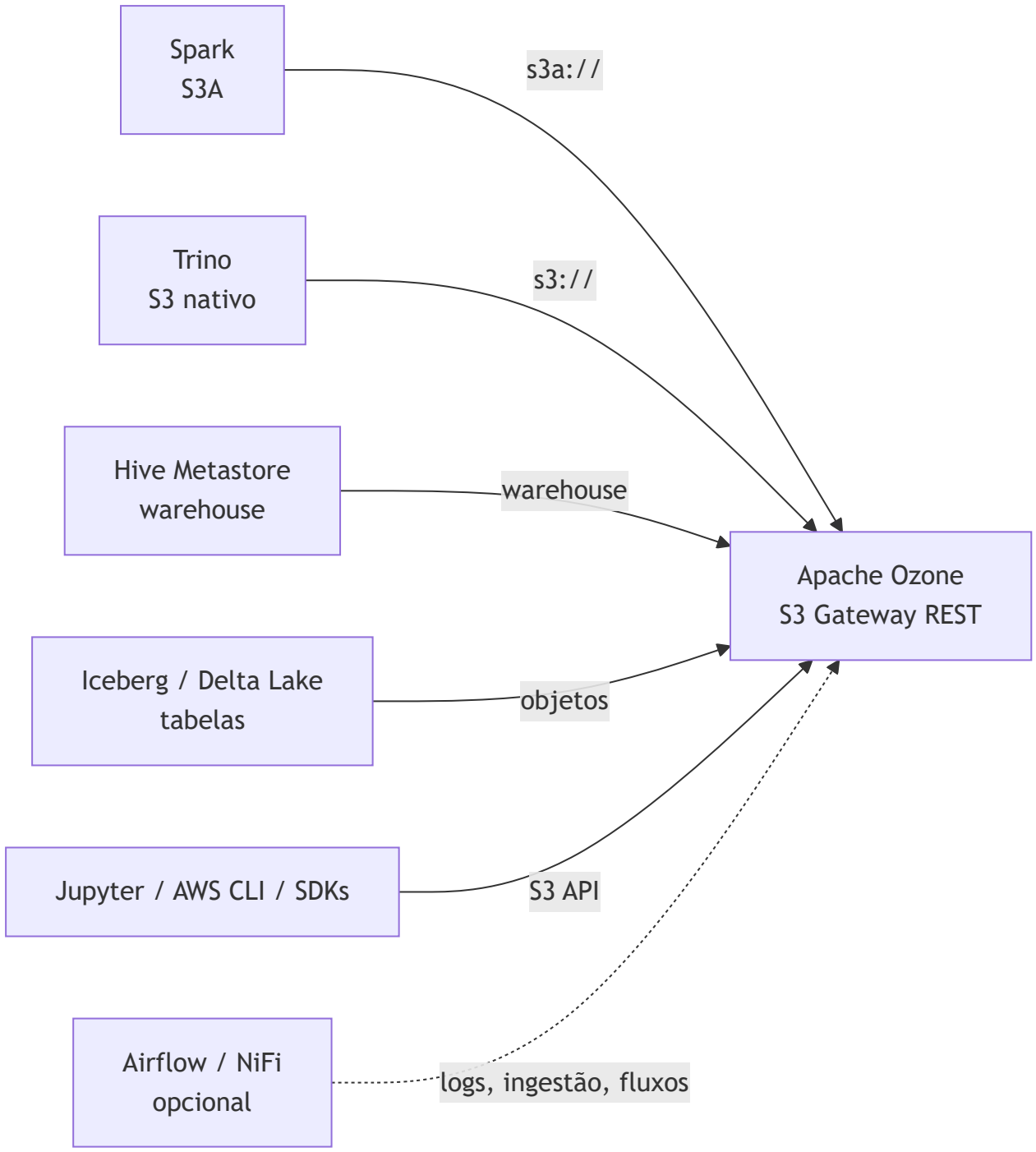
Version 3.0.1 | Type application | AppVersion 2.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações

© Copyright Tecnisys 2026. All rights reserved.



O Ozone fornece armazenamento de objetos compatível com S3 que pode substituir MinIO ou AWS S3 na stack TDP. A tabela abaixo resume os componentes que podem ser integrados, com indicação de prioridade e complexidade:

© Copyright Tecnisys 2026. All rights reserved.

Componente	Tipo de integração	Prioridade	Complexidade
Spark	Armazenamento primário para jobs	★★★ Alta	Baixa
Trino	Backend de catá-logo Hive/Iceberg	★★★ Alta	Baixa
Hive Metastore	Warehouse de metadados	★★★ Alta	Baixa
Delta Lake	Armazenamento e manutenção de tabelas	★★★ Alta	Baixa
Iceberg	Armazenamento de formato de tabela	★★★ Alta	Baixa
Airflow	DAGs, logs e XCom backend	★★ Média	Média
NiFi	Fluxos de dados e repositório de conteúdo	★★ Média	Média
Jupyter	Acesso a dados e notebooks	★★ Média	Baixa
Superset	Cache de resulta-dos de consulta	★ Baixa	Baixa
Ranger	Armazenamento de logs de auditoria	★ Baixa	Média

© Copyright Tecnisys 2026. All rights reserved.

Endpoint interno padrão do S3 Gateway REST:

```
http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878
```

Use Ingress apenas quando o consumidor estiver fora do cluster ou quando a arquitetura exigir um hostname externo, como `https://<OZONE_S3_REST_HOSTNAME>`.

> **SEGURANÇA DO ENDPOINT S3**
>
> O S3 Gateway REST é o plano de dados do Ozone. Antes de expô-lo ou configurar consumidores, garanta autenticação habilitada, TLS para tráfego fora do cluster e controle de rede adequado. Consulte Segurança - Apache Ozone.

Credenciais e endpoint

Os exemplos abaixo usam AWS Signature v4 em modo simples. As credenciais devem vir de Secrets Kubernetes, variáveis de ambiente seguras ou mecanismo equivalente do componente consumidor. Não versionar `<AWS_ACCESS_KEY_ID>` nem `<AWS_SECRET_ACCESS_KEY>` em arquivos Git.

Para Ozone, mantenha path-style access habilitado nos consumidores S3, pois o bucket normalmente faz parte do caminho da URL.

Para detalhes sobre modelo de credenciais, rotação, TLS e controle de rede, consulte a página de segurança do Ozone.

Spark com S3A

Por que integrar

- Armazenar dados de entrada e saída de jobs Spark
- Persistir DataFrames e RDDs
- Compartilhar dados entre aplicações Spark
- Habilitar arquiteturas de data lake

Configuração

© Copyright Tecnisys 2026. All rights reserved.

Propriedades mínimas de conexão:

```
spark.hadoop.fs.s3a.endpoint=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
spark.hadoop.fs.s3a.access.key=<AWS_ACCESS_KEY_ID>
spark.hadoop.fs.s3a.secret.key=<AWS_SECRET_ACCESS_KEY>
spark.hadoop.fs.s3a.path.style.access=true
spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem
spark.hadoop.fs.s3a.connection.ssl.enabled=false
```

Para volumes altos de dados, adicione as propriedades de tuning ao mesmo bloco:

```
spark.hadoop.fs.s3a.connection.maximum=100
spark.hadoop.fs.s3a.threads.max=64
spark.hadoop.fs.s3a.fast.upload=true
spark.hadoop.fs.s3a.block.size=128M
```

Em implantações Kubernetes, entregue as propriedades via ConfigMap. Crie o manifesto:

spark-ozone-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: spark-custom-defaults
  namespace: <NAMESPACE>
data:
  spark-defaults.conf: |
    spark.hadoop.fs.s3a.endpoint=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
    spark.hadoop.fs.s3a.access.key=<AWS_ACCESS_KEY_ID>
    spark.hadoop.fs.s3a.secret.key=<AWS_SECRET_ACCESS_KEY>
    spark.hadoop.fs.s3a.path.style.access=true
    spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem
    spark.hadoop.fs.s3a.connection.ssl.enabled=false
    spark.hadoop.fs.s3a.connection.maximum=100
    spark.hadoop.fs.s3a.threads.max=64
```

© Copyright Tecnisys 2026. All rights reserved.

```
    spark.hadoop.fs.s3a.fast.upload=true
    spark.hadoop.fs.s3a.block.size=128M
```

Aplique o ConfigMap e referencie-o no `values.yaml` do chart Spark:

Terminal input

```
kubectl apply -f spark-ozone-config.yaml -n <NAMESPACE>
```

values.yaml (trecho)

```
spark:
  master:
    existingConfigmap: spark-custom-defaults
  worker:
    existingConfigmap: spark-custom-defaults
```

Exemplo de uso

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Ozone S3 Example") \
    .getOrCreate()

# Ler do Ozone
df = spark.read.parquet("s3a://warehouse/data/input.parquet")

# Processar dados
result = df.groupBy("category").count()

# Escrever no Ozone
result.write.mode("overwrite").parquet("s3a://warehouse/data/output.parquet")
```

Criação do Secret Kubernetes

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl create secret generic spark-ozone-credentials \
  --from-literal=access-key=<AWS_ACCESS_KEY_ID> \
  --from-literal=secret-key=<AWS_SECRET_ACCESS_KEY> \
  -n <NAMESPACE>
```

Trino com S3 nativo

Por que integrar

- Consultar dados armazenados no Ozone via catálogos Hive/Iceberg
- Queries federadas entre múltiplas fontes de dados
- Analytics de alta performance em armazenamento de objetos

Configuração

Configure os catálogos no `values.yaml` do chart Trino:

values.yaml (trecho)

```
tdp-trino:
  catalogs:
    hive: |
      connector.name=hive
      hive.metastore.uri=thrift://tdp-hive-metastore.
<NAMESPACE>.svc.cluster.local:9083
      hive.non-managed-table-writes-enabled=true
      hive.non-managed-table-creates-enabled=true
      fs.native-s3.enabled=true
      s3.endpoint=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
      s3.region=us-east-1
      s3.aws-access-key=<AWS_ACCESS_KEY_ID>
      s3.aws-secret-key=<AWS_SECRET_ACCESS_KEY>
      s3.path-style-access=true

    iceberg: |
      connector.name=iceberg
      iceberg.catalog.type=hive_metastore
      hive.metastore.uri=thrift://tdp-hive-metastore.
<NAMESPACE>.svc.cluster.local:9083
      fs.native-s3.enabled=true
```

© Copyright Tecnisys 2026. All rights reserved.

```
      s3.endpoint=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
      s3.region=us-east-1
      s3.aws-access-key=<AWS_ACCESS_KEY_ID>
      s3.aws-secret-key=<AWS_SECRET_ACCESS_KEY>
      s3.path-style-access=true
```

Ajuste nomes de Services, catálogos e Secrets conforme os releases reais do ambiente.

Exemplo de uso

```
-- Criar schema apontando para bucket Ozone
CREATE SCHEMA iceberg.warehouse WITH (location = 's3a://warehouse/');

-- Criar tabela Iceberg
CREATE TABLE iceberg.warehouse.vendas (
    id      BIGINT,
    produto VARCHAR,
    valor   DECIMAL(10,2),
    dt_venda DATE
) WITH (format = 'PARQUET', location = 's3a://warehouse/vendas');

-- Inserir dados
INSERT INTO iceberg.warehouse.vendas VALUES
    (1, 'Laptop', 1200.00, DATE '2024-01-15'),
    (2, 'Mouse',    25.50, DATE '2024-01-16');

-- Consultar
SELECT produto, SUM(valor) AS total FROM iceberg.warehouse.vendas GROUP BY
produto;
```

Hive Metastore e warehouse

Por que integrar

- Armazenar metadados de tabelas e dados do warehouse no Ozone
- Repositório central de metadados para Spark, Trino e demais ferramentas

Configuração

Configure o Hive Metastore para usar o Ozone como warehouse no `values.yaml`:

© Copyright Tecnisys 2026. All rights reserved.

values.yaml (trecho)

```
tdp-hive-metastore:
  metastore:
    type: s3
    s3:
      endpoint: http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
      accessKey: <AWS_ACCESS_KEY_ID>
      secretKey: <AWS_SECRET_ACCESS_KEY>
      bucket: warehouse
      pathStyleAccess: true
  warehouse:
    dir: s3a://warehouse/hive
```

Quando Spark e Trino compartilham o mesmo warehouse, mantenha endpoint, região, estilo de acesso e credenciais alinhados nos dois lados.

Iceberg e Delta Lake

Por que integrar

- Armazenar tabelas Iceberg e Delta Lake no Ozone
- Transações ACID em armazenamento de objetos
- Time travel e versionamento de dados

Configuração

Exemplos de localizações:

```
s3a://warehouse/iceberg
s3a://warehouse/delta
```

Prefira buckets e prefixos separados por domínio ou ambiente.

Para jobs de manutenção Delta Lake (VACUUM, OPTIMIZE), configure o `values.yaml` do chart Delta Lake:

© Copyright Tecnisys 2026. All rights reserved.

values.yaml (trecho)

```
maintenance:
  spark:
    config:
      "spark.sql.extensions": "io.delta.sql.DeltaSparkSessionExtension"
      "spark.sql.catalog.spark_catalog":
"org.apache.spark.sql.delta.catalog.DeltaCatalog"
      "spark.hadoop.fs.s3a.endpoint": "http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878"
      "spark.hadoop.fs.s3a.path.style.access": "true"
      "spark.hadoop.fs.s3a.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem"
      "spark.hadoop.fs.s3a.connection.ssl.enabled": "false"
      "spark.databricks.delta.retentionDurationCheck.enabled": "false"

  defaultTablePath: "s3a://warehouse/delta"
```

Para executar VACUUM diretamente via `spark-sql`:

Terminal input

```
spark-sql \
  --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \
  --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
\
  --conf spark.hadoop.fs.s3a.endpoint=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878 \
  --conf spark.hadoop.fs.s3a.path.style.access=true \
  --conf spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem \
  --conf spark.hadoop.fs.s3a.access.key=<AWS_ACCESS_KEY_ID> \
  --conf spark.hadoop.fs.s3a.secret.key=<AWS_SECRET_ACCESS_KEY> \
  --conf spark.databricks.delta.retentionDurationCheck.enabled=false \
  -e "VACUUM delta.\`s3a://warehouse/delta/vendas\` RETAIN 168 HOURS;"
```

A política de retenção, limpeza e manutenção continua sendo definida no componente consumidor.

Jupyter e clientes S3

© Copyright Tecnisys 2026. All rights reserved.



Por que integrar

- Acessar dados do Ozone em notebooks de análise
- Compartilhar datasets entre usuários
- Armazenar notebooks e resultados de análise

Configuração via values.yaml

Injete as credenciais como variáveis de ambiente no chart JupyterHub:

values.yaml (trecho)

```
tdp-jupyter:
  extraEnv:
    - name: AWS_ACCESS_KEY_ID
      value: "<AWS_ACCESS_KEY_ID>"
    - name: AWS_SECRET_ACCESS_KEY
      value: "<AWS_SECRET_ACCESS_KEY>"
    - name: AWS_ENDPOINT_URL
      value: "http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878"
    - name: AWS_DEFAULT_REGION
      value: "us-east-1"
```

> **TIP**
>
> Em produção, prefira referenciar as credenciais a partir de um Secret Kubernetes em vez de valores literais no `values.yaml`.

Exemplos em notebooks

Usando boto3 com pandas:

```
import boto3
import pandas as pd

s3 = boto3.client(
    "s3",
    endpoint_url="http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878",
    aws_access_key_id="<AWS_ACCESS_KEY_ID>",
    aws_secret_access_key="<AWS_SECRET_ACCESS_KEY>",
```

© Copyright Tecnisys 2026. All rights reserved.

```
    region_name="us-east-1",
)

# Ler dados do Ozone
obj = s3.get_object(Bucket="notebooks", Key="data/vendas.csv")
df = pd.read_csv(obj["Body"])

# Processar
resultado = df.groupby("categoria")["valor"].sum()

# Gravar resultado de volta no Ozone
resultado.to_csv("/tmp/resultado.csv")
s3.upload_file("/tmp/resultado.csv", "notebooks", "data/resultado.csv")
```

Usando AWS CLI:

Terminal input

```
aws configure set aws_access_key_id <AWS_ACCESS_KEY_ID>
aws configure set aws_secret_access_key <AWS_SECRET_ACCESS_KEY>
aws configure set region us-east-1
aws s3 ls --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
```

Airflow e NiFi

Airflow

Por que integrar

- Armazenar logs remotos no Ozone
- Usar Ozone como backend de XCom para passagem de dados volumosos
- Armazenar artefatos de DAGs

Configuração

Configure logs remotos e a conexão AWS no `values.yaml` do chart Airflow:

values.yaml (trecho)

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-airflow:
  config:
    logging:
      remote_logging: "True"
      remote_base_log_folder: "s3://airflow-logs/"
      remote_log_conn_id: "ozone_s3"

  connections:
    - conn_id: ozone_s3
      conn_type: aws
      conn_extra: |
        {
          "aws_access_key_id": "<AWS_ACCESS_KEY_ID>",
          "aws_secret_access_key": "<AWS_SECRET_ACCESS_KEY>",
          "endpoint_url": "http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878",
          "region_name": "us-east-1"
        }
```

Exemplo em DAGs

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.amazon.aws.hooks.s3 import S3Hook
from datetime import datetime

with DAG("ozone_example", start_date=datetime(2024, 1, 1),
schedule_interval=None) as dag:

    def upload_to_ozone():
        s3_hook = S3Hook(aws_conn_id="ozone_s3")
        s3_hook.load_file(
            filename="/tmp/dados.csv",
            key="dados/entrada.csv",
            bucket_name="airflow-data",
        )

    upload_task = PythonOperator(
        task_id="upload_to_ozone",
        python_callable=upload_to_ozone,
    )
```

© Copyright Tecnisys 2026. All rights reserved.



NiFi

Por que integrar

- Ingerir dados diretamente no Ozone
- Processar e transformar dados armazenados no Ozone
- Armazenar conteúdo de flowfiles

Configuração

Use os processadores `PutS3Object` e `FetchS3Object` com as seguintes configurações:

PutS3Object:

- **Endpoint Override URL**: `http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878`
- **Access Key** e **Secret Key**: credenciais criadas para o serviço NiFi
- **Region**: `us-east-1`
- **Bucket**: `nifi-data`
- **Path Style Access**: habilitado

FetchS3Object: mesmas configurações de endpoint, credenciais e região.

Exemplo de fluxo simples: `GenerateFlowFile → PutS3Object (Ozone) → LogAttribute`

Organização de buckets por componente

No armazenamento de objetos compatível com S3, um **bucket** é a unidade básica de organização: funciona como um contêiner de nível superior onde os objetos (arquivos, dados) são armazenados, cada um identificado por uma chave única dentro do bucket.

Cada consumidor deve ter seu próprio bucket (ou prefixo dedicado) e credenciais independentes. Isso simplifica o controle de acesso e permite revogar credenciais por serviço sem impactar os demais.

Sugestão de estrutura para um ambiente TDP completo:

© Copyright Tecnisys 2026. All rights reserved.

Bucket	Uso principal
`warehouse`	Tabelas Spark, Hive Metastore, Iceberg e Delta Lake
`airflow-logs`	Logs remotos do Airflow
`airflow-data`	Artefatos e dados de DAGs
`nifi-data`	Dados ingeridos ou processados pelo NiFi
`notebooks`	Notebooks e dados do JupyterHub

Crie os buckets com o AWS CLI apontado para o endpoint interno do Ozone:

Terminal input

```
aws s3 mb s3://warehouse --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
aws s3 mb s3://airflow-logs --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
aws s3 mb s3://nifi-data --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
aws s3 mb s3://notebooks --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878
```

Sugestão de organização interna dos buckets:

```
warehouse/
├── hive/        # Tabelas Hive
├── spark/       # Saída de jobs Spark
├── delta/       # Tabelas Delta Lake
└── iceberg/     # Tabelas Iceberg

airflow-logs/
└── dag_id=<DAG_ID>/
    └── run_id=<RUN_ID>/
```

© Copyright Tecnisys 2026. All rights reserved.

```
nifi-data/
├── input/       # Dados brutos
├── processing/  # Dados em processamento
└── output/      # Dados processados

notebooks/
├── users/       # Notebooks por usuário
└── shared/      # Notebooks compartilhados
```

Para criar credenciais independentes por componente, use o script disponível no chart Ozone (veja a página de segurança):

Terminal input

```
./scripts/generate-s3-credentials.sh create spark-service
./scripts/generate-s3-credentials.sh create trino-service
./scripts/generate-s3-credentials.sh create airflow-service
./scripts/generate-s3-credentials.sh create jupyter-service
```

Teste rápido com dados de exemplo

Para validar a integração S3 logo após a instalação, crie buckets de teste e envie um arquivo de exemplo:

Terminal input

```
# Bucket para cargas de teste do Spark
aws s3 mb s3://spark-data --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878

# Bucket para testes de performance
aws s3 mb s3://performance-test --endpoint-url=http://<RELEASE_NAME>-s3g-rest.
<NAMESPACE>.svc.cluster.local:9878

# Envie um arquivo de exemplo para o bucket spark-data
aws s3 cp produtos.csv s3://spark-data/ --endpoint-url=http://<RELEASE_NAME>-
s3g-rest.<NAMESPACE>.svc.cluster.local:9878
```

© Copyright Tecnisys 2026. All rights reserved.

Conteúdo de exemplo para `produtos.csv`:

```
id,nome,valor
1,produtoA,100
2,produtoB,250
3,produtoC,45
```

> **SEGURANÇA**
>
> Para práticas de segurança (uso de Secrets, rotação de credenciais, TLS, NetworkPolicy e auditoria), consulte a página de segurança do Ozone.

Próximos passos

Recomendação de ordem de integração:

1. **Spark + Hive Metastore** — base do data lake
2. **Trino** — motor de consulta analítica
3. **Airflow** — orquestração
4. **Ferramentas especializadas** (NiFi, Jupyter) conforme necessidade

---

Solução de problemas

Para verificar a conectividade ao endpoint S3 a partir de um pod consumidor:

Terminal input

```
kubectl exec -it <POD_NAME> -n <NAMESPACE> -- curl http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878
kubectl exec -it <POD_NAME> -n <NAMESPACE> -- aws s3 ls --endpoint-url=http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878
```

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Causa provável	Correção
`Name or service not known`	DNS interno ou namespace incorreto	Verifique `http://<RELEASE_NAME>-s3g-rest.<NAMESPACE>.svc.cluster.local:9878` e o namespace do release
Conexão recusada ou timeout	Service sem endpoints, NetworkPolicy ou porta errada	Verifique pods S3 Gateway, endpoints do Service e políticas de rede
Erro de bucket virtual-hosted-style	Cliente tentando usar bucket no hostname	Habilite path-style access no consumidor
`SignatureDoesNotMatch` ou 403	Credenciais, região, endpoint ou relógio inconsistentes	Confira Secret, região `us-east-1`, URL exata e sincronização de horário
Erro TLS	Cliente não confia no certificado ou usa HTTP/HTTPS divergente	Alinhe URL, certificado, truststore e configuração TLS do consumidor

© Copyright Tecnisys 2026. All rights reserved.

PostgreSQL

CHART

Version 3.0.1 | Type application | AppVersion 17.5.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

O que é o PostgreSQL no TDP?

O PostgreSQL é o banco de dados relacional da infraestrutura TDP Kubernetes. Ele não é o motor analítico da plataforma; seu papel é oferecer armazenamento transacional para configurações, estado interno, catálogos, usuários, permissões ou histórico operacional dos componentes que o consomem.

Esta página concentra o próprio PostgreSQL: instalação, persistência, recursos, parâmetros de `postgresql.conf`, Service interno e credenciais administrativas. A configuração dos componentes que consomem esse banco deve ser feita na página de cada componente, porque cada integração define seus próprios parâmetros, Secrets e validações.

Para ambientes que exigirem alta disponibilidade, backup corporativo, replicação, políticas de manutenção próprias ou isolamento por componente, avalie usar uma instância PostgreSQL dedicada ou externa.

Estrutura de valores (Helm)

Todos os valores do PostgreSQL devem ficar sob a chave **`tdp-postgresql:`** (alias da dependência no `Chart.yaml`).

```
tdp-postgresql:
  primary:
    persistence:
      size: 10Gi
```

Visão geral

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Chart	`tdp-postgresql`
Versão do PostgreSQL	17.5.0
Versão do Chart	3.0.1
Registry (OCI)	`oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql`

Páginas relacionadas

- Segurança - Padrões Gerais: credenciais, Secrets e boas práticas aplicáveis ao PostgreSQL.
- Integrações - Padrões Gerais: padrão de banco externo usado pelos componentes que consomem este PostgreSQL.

Pré-requisitos

Antes de instalar, confirme:

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+;
- Helm 3.2.0+;
- StorageClass disponível para volumes persistentes;
- namespace definido para o release.

Em OpenShift, a compatibilidade de Security Context é habilitada por padrão com `tdp-postgresql.global.compatibility.openshift.adaptSecurityContext: auto` e `tdp-postgresql.volumePermissions.enabled: false`. Em Kubernetes vanilla, a escrita no volume ainda depende das permissões fornecidas pela StorageClass e pelo PersistentVolume do cluster.

Instalação

Instalação

© Copyright Tecnisys 2026. All rights reserved.



O mesmo comando de instalação é usado em Kubernetes vanilla e OpenShift. Em OpenShift, a compatibilidade de Security Context já é tratada pelos valores padrão do chart.

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql -n <NAMESPACE> --
create-namespace
```

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão
`tdp-postgresql.postgresql.enabled`	Habilita o subchart PostgreSQL	`true`
`tdp-postgresql.primary.persistence.enabled`	Habilita persistência	`true`
`tdp-postgresql.primary.persistence.size`	Tamanho do PVC	`1Gi`
`tdp-postgresql.primary.resources.requests.cpu`	CPU solicitada	`500m`
`tdp-postgresql.primary.resources.requests.memory`	Memória solicitada	`1Gi`
`tdp-postgresql.primary.resources.limits.cpu`	Limite de CPU	`1000m`
`tdp-postgresql.primary.resources.limits.memory`	Limite de memória	`2Gi`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão
`tdp-postgresql.primary.configuration`	Sobrescritas de `postgresql.conf`	`""`
`tdp-postgresql.global.compatibility.openshift.adaptSecurityContext`	Modo de compatibilidade com OpenShift SCC	`auto`
`tdp-postgresql.volumePermissions.enabled`	Init container de permissões de volume	`false`
`tdp-postgresql.service.type`	Tipo do Service	`ClusterIP`
`tdp-postgresql.service.port`	Porta do Service	`5432`
`tdp-postgresql.startupProbe.enabled`	Habilita a probe de inicialização do PostgreSQL	`true`
`tdp-postgresql.metrics.enabled`	Exporter de métricas	`false`

Para consultar a lista completa de parâmetros da versão instalada:

Terminal input

```
helm show values oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql --
version <CHART_VERSION>
```

Detalhes de configuração

Persistência e recursos

© Copyright Tecnisys 2026. All rights reserved.



O PostgreSQL armazena seus dados em PVC quando `tdp-postgresql.primary.persistence.enabled: true`. Ajuste `tdp-postgresql.primary.persistence.size` conforme o volume esperado de metadados, retenção operacional e política de backup do ambiente.

Os recursos de CPU e memória ficam em `tdp-postgresql.primary.resources.*`. Dimensione esses valores considerando número de conexões, carga dos componentes consumidores, frequência de jobs administrativos e janelas de manutenção.

Sobrescritas em `postgresql.conf`

O chart aceita configuração customizada via `tdp-postgresql.primary.configuration`. Use esse bloco quando precisar ajustar parâmetros do PostgreSQL, como conexões, WAL, logs ou comportamento de escrita.

Exemplo:

```
tdp-postgresql:
  primary:
    configuration: |-
      listen_addresses = '*'
      max_connections = 400
      superuser_reserved_connections = 10
      wal_level = 'replica'
      fsync = 'on'
      max_wal_size = '400MB'
      max_wal_senders = '16'
      wal_keep_size = '128MB'
      hot_standby = 'on'
      log_connections = 'false'
      log_disconnections = 'false'
      log_hostname = 'false'
      client_min_messages = 'error'
```

Para usar o comportamento padrão do subchart, mantenha `tdp-postgresql.primary.configuration: ""` ou omita o bloco.

Acesso & Segurança

Acesso

© Copyright Tecnisys 2026. All rights reserved.

Por padrão, o PostgreSQL é exposto por um Service `ClusterIP` na porta `5432`. Esse acesso é destinado aos componentes e clientes que executam dentro do cluster:

```
<RELEASE_NAME>.<NAMESPACE>.svc.cluster.local:5432
```

Obter a senha do usuário `postgres`

Terminal input

```
export POSTGRES_PASSWORD=$(kubectl get secret --namespace <NAMESPACE>
<RELEASE_NAME> -o jsonpath="{.data['postgres-password']}" | base64 -d)
```

Conectar a partir de um pod temporário

Para validar o acesso sem instalar cliente PostgreSQL na máquina local, crie um pod temporário com a imagem cliente e conecte-se com o `psql`. O pod se autoexclui ao encerrar a sessão (`--rm`):

Terminal input

```
kubectl run <RELEASE_NAME>-client --rm --tty -i --restart='Never' --namespace
<NAMESPACE> \
  --image docker.io/bitnamilegacy/postgresql:17.5.0-debian-12-r20 --
env="PGPASSWORD=$POSTGRES_PASSWORD" \
  --command -- psql -h <RELEASE_NAME> -p 5432 -U postgres -d postgres
```

Acesso pontual fora do cluster

Use `port-forward` apenas para administração ou diagnóstico pontual:

Terminal input

```
kubectl port-forward --namespace <NAMESPACE> svc/<RELEASE_NAME> 5432:5432
```

Em outro terminal, conecte com `psql`:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
psql -h 127.0.0.1 -p 5432 -U postgres -d postgres
```

Segurança e credenciais

A senha do usuário `postgres` é armazenada em um Kubernetes Secret do release, na chave `postgres-password`. Trate esse Secret como credencial administrativa.

Não versione senhas em texto plano em arquivos de valores. Quando outro componente precisar criar ou acessar bancos neste PostgreSQL, configure a referência ao Secret conforme a documentação desse componente.

Para boas práticas gerais sobre Secrets, autenticação e TLS, consulte Segurança - Padrões Gerais.

Solução de problemas

Solução de problemas

Após instalar ou atualizar, valide os recursos principais:

Terminal input

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get svc
kubectl -n <NAMESPACE> get pvc
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

Problemas comuns:

Sintoma	Possível causa	Verificação
Pod não inicia	PVC pendente ou StorageClass indisponível	`kubectl -n <NAMESPACE> get pvc`

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Possível causa	Verificação
Falha de escrita no volume	Permissões do PV/StorageClass incompatíveis	Eventos do pod e política do StorageClass
Componentes não conectam	Service, Secret ou senha incorretos	DNS do Service, Secret do release e logs do componente
Conexões saturadas	`max_connections` insuficiente para a carga	`pg_stat_activity` e parâmetros em `postgresql.conf`

Desinstalação

Desinstalação

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

PostgreSQL – Integração

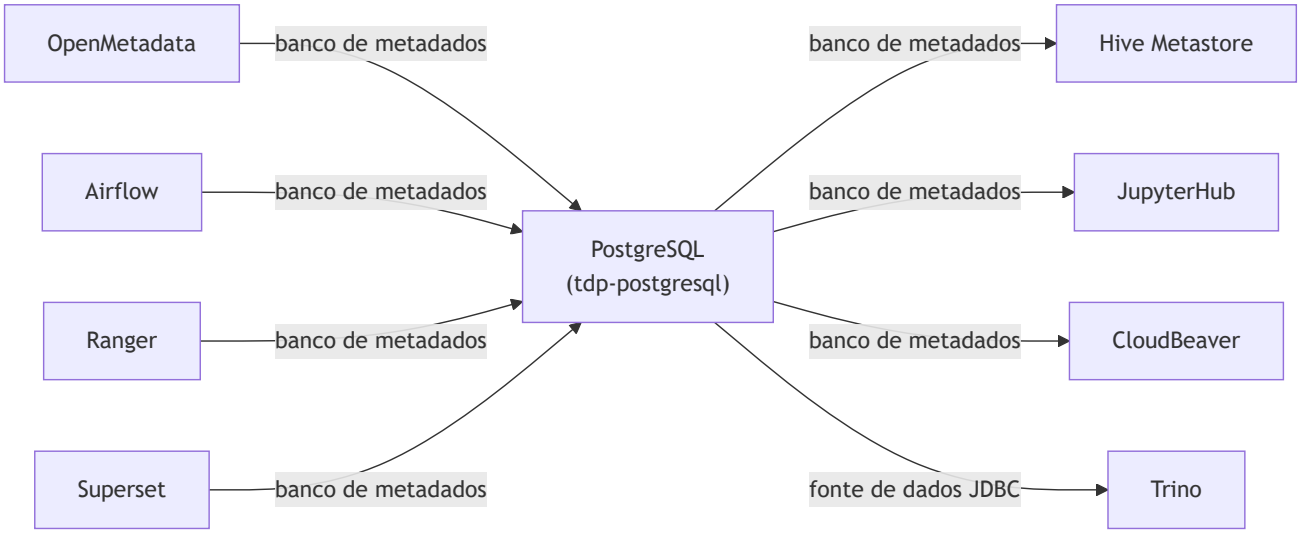
CHART

Version 3.0.1 | Type application | AppVersion 17.5.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral



O chart `tdp-postgresql` disponibiliza um servidor PostgreSQL interno ao cluster. Charts como OpenMetadata, Airflow, Ranger, Superset, Hive Metastore e JupyterHub podem usá-lo como banco de metadados externo via `TDP-Settings.externalDatabase`. O CloudBeaver usa PostgreSQL como banco de metadados externo (via `tdp-cloudbeaver.database.*`). O Trino pode consultá-lo como fonte de dados via conector JDBC.

A integração com cada componente é descrita na página de integrações do respectivo chart:

- **OpenMetadata**: consulte Configuração do OpenMetadata — seção *Banco de dados (PostgreSQL)*.
- **Airflow**: consulte Configuração do Airflow — seção *Configuração do banco de dados*.

© Copyright Tecnisys 2026. All rights reserved.



- **Ranger**: consulte Configuração do Ranger — seção *Banco de dados e auditoria*.
- **Superset**: consulte Configuração do Superset — seção *PostgreSQL de metadados*.
- **Hive Metastore**: consulte Configuração do Hive Metastore — seção *Banco de dados*.
- **JupyterHub**: consulte Configuração do Jupyter — seção *Banco de dados*.
- **CloudBeaver**: consulte Configuração do CloudBeaver — seção *Banco de dados*.
- **Trino**: consulte Integrações — Trino — seção *PostgreSQL*.

Endereço do serviço interno

Depois de instalado, o PostgreSQL fica acessível dentro do cluster pelo nome DNS:

```
<RELEASE_NAME>.<NAMESPACE>.svc.cluster.local:5432
```

Substitua `<RELEASE_NAME>` pelo nome do release Helm e `<NAMESPACE>` pelo namespace onde o chart foi instalado.

Obter a senha do admin

A senha do usuário `postgres` é gerada automaticamente pelo chart e armazenada em um Kubernetes Secret:

```
export POSTGRES_PASSWORD=$(kubectl get secret --namespace <NAMESPACE>
<RELEASE_NAME> \
  -o jsonpath="{.data['postgres-password']}" | base64 -d)
```

> **INFO**
>
> O nome do Secret corresponde ao nome do release (ex.: `tdp-postgresql`), com a chave `postgres-password`.

Conectar a partir de dentro do cluster

Para testes ou diagnósticos a partir de um pod temporário no mesmo namespace:

```
kubectl run <RELEASE_NAME>-client --rm --tty -i --restart='Never' \
  --namespace <NAMESPACE> \
```

© Copyright Tecnisys 2026. All rights reserved.

```
  --image docker.io/bitnamilegacy/postgresql:17.5.0-debian-12-r20 \
  --env="PGPASSWORD=$POSTGRES_PASSWORD" \
  --command -- psql -h <RELEASE_NAME> -p 5432 -U postgres -d postgres
```

Conectar a partir de fora do cluster (port-forward)

Para acesso temporário de fora do cluster — por exemplo, de uma estação de trabalho com o cliente `psql`:

```
kubectl port-forward --namespace <NAMESPACE> svc/<RELEASE_NAME> 5432:5432
psql -h 127.0.0.1 -p 5432 -U postgres -d postgres
```

> **NOTE**
>
> O `port-forward` é adequado para testes e diagnóstico. Não é um método de acesso permanente.

Validação

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get svc
kubectl -n <NAMESPACE> get pvc
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

© Copyright Tecnisys 2026. All rights reserved.

Ranger

CHART

Version 3.0.1 | Type application | AppVersion 2.7.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

O que é o Apache Ranger no TDP?

O Apache Ranger é um framework de segurança centralizado para plataformas de dados.
Em vez de cada serviço gerenciar suas próprias regras de acesso, o Ranger fornece um ponto único onde você define **quem pode fazer o quê em qual recurso**.
Os serviços integrados consultam essas políticas para permitir ou negar ações em tempo de execução.

No TDP Kubernetes, o Ranger atua como ponto central de administração de políticas de acesso para os serviços que possuem integração habilitada.

Cada integração usa o mecanismo de autorização compatível com o respectivo serviço para consultar o Ranger Admin e aplicar as decisões de acesso conforme as políticas configuradas.

> **PARA SABER MAIS**
>
> Consulte Apache Ranger - Conceitos para uma visão completa da ferramenta, seu modelo de plugins e funcionamento.

Estrutura de valores (Helm)

O chart `tdp-ranger` usa três blocos principais de valores:

- `tdp-ranger:` — configuração do Ranger Admin: recursos, credenciais, banco de dados, Solr, OpenShift.

© Copyright Tecnisys 2026. All rights reserved.



- `rangerIntegrations:` — automação de cadastro de serviços no Ranger Admin e políticas declarativas.
- `TDP-Settings:` — helpers TDP para banco externo e exposição HTTP/HTTPS.

```
tdp-ranger:
  ranger:
    adminUser: admin
    database:
      host: "<POSTGRESQL_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: 5432
```

Arquitetura e componentes

Componente	Função
Ranger Admin	Interface e API central para cadastro de serviços, políticas e auditoria
PostgreSQL	Armazena metadados do Ranger, como políticas, usuários e configuração
Solr	Armazena logs de auditoria quando habilitado pelo subchart
Plugins e integrações	Permitem que serviços configurados consultem o Ranger para autorização
Job de integrações	Automatiza o cadastro no Ranger Admin dos serviços habilitados em `rangerIntegrations` e das políticas declaradas em `defaultPolicies`, quando configuradas

O Ranger Admin é o ponto central de administração.
Os serviços integrados continuam executando suas cargas normalmente, mas consultam as políticas cadastradas no Ranger para decidir se uma ação deve ser permitida.

© Copyright Tecnisys 2026. All rights reserved.

Visão geral

Propriedade	Valor
Chart	`tdp-ranger`
Versão do Ranger	2.7.0
Versão do Chart	3.0.1
AppVersion do chart	`2.7.0`
Registry	`oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger`

Páginas relacionadas

- Segurança - Ranger: credenciais, LDAP/AD, UserSync, LDAPS e boas práticas.
- Integrações - Ranger: automação de cadastro no Ranger Admin, exemplos de integrações e políticas declaradas em `defaultPolicies`.
- Exposição externa - Ranger: exposição do Ranger Admin por Ingress ou Gateway API.

Pré-requisitos

Antes de instalar ou customizar o chart `tdp-ranger`, confirme:

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível para volumes persistentes.
- Registry OCI da Tecnisys acessível pelo ambiente de instalação.
- Senha forte definida para o usuário administrador do Ranger.
- Definição prévia de banco PostgreSQL interno ou externo para metadados.
- Quando usar integrações: serviços de destino implantados, acessíveis por rede e configurados para usar o mecanismo de autorização compatível.

Instalação

© Copyright Tecnisys 2026. All rights reserved.

Instalação

Exemplo

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger
-n <NAMESPACE> --create-namespace --timeout 10m
```

Ajuste credenciais, banco de dados, persistência do Solr, exposição de rede e integrações pelo seu arquivo de valores.

> **OPENSHIFT**
>
> Em OpenShift, habilite `tdp-ranger.openshift.enabled=true` para criar o RoleBinding que associa a ServiceAccount do Ranger à SCC `anyuid`.
> Essa configuração é necessária porque o container do Ranger executa scripts que exigem permissões elevadas.

Exemplo OpenShift

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger
-n <NAMESPACE> --create-namespace --timeout 10m --set tdp-
ranger.openshift.enabled=true
```

Valide a criação do RoleBinding no namespace:

```
oc get rolebinding -n <NAMESPACE> | grep anyuid
oc describe rolebinding <RELEASE_NAME>-openshift-anyuid-scc -n <NAMESPACE>
```

Parâmetros principais

Parâmetros principais

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`tdp-ranger.enabled`	Habilitar o deploy do Ranger	`true`
`tdp-ranger.clusterLabel`	Label de cluster	`tdp`
`tdp-ranger.ranger.adminUser`	Usuário admin Ranger	`admin`
`tdp-ranger.ranger.adminPassword`	Senha admin Ranger	`ChangeMe!2026` (exemplo — alterar antes de produção)
`tdp-ranger.ranger.dbUser`	Usuário do banco	`ranger`
`tdp-ranger.ranger.dbPassword`	Senha do banco	`ChangeMe!2026` (exemplo — alterar antes de produção)
`tdp-ranger.ranger.existingSecret`	Secret existente para credenciais do DB	`<RELEASE_NAME>-ranger-database`
`tdp-ranger.ranger.existingSecretPasswordKey`	Chave da senha no Secret	`password`
`tdp-ranger.ranger.database.host`	Host do banco	`<POSTGRESQL_SERVICE>.<NAMESPACE>.svc.cluster.local`
`tdp-ranger.ranger.database.port`	Porta do banco	`5432`
`TDP-Settings.externalDatabase.enabled`	Habilitar job de bootstrap do DB externo	`false`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`TDP-Settings.externalDatabase.recreate`	Recriar DB em install/upgrade	`true`
`TDP-Settings.externalDatabase.externalSecret.releaseName`	Nome do release PostgreSQL	`tdp-postgresql`
`rangerIntegrations.configJob.enabled`	Habilitar job de integrações	`false`
`rangerIntegrations.configJob.image`	Imagem do job	`registry.tecnisys.com.br/community/images/python:3.11-slim`
`rangerIntegrations.configJob.rangerReadyTimeout`	Timeout de prontidão, em segundos	`600`
`rangerIntegrations.configJob.retryInterval`	Intervalo de retry, em segundos	`10`
`rangerIntegrations.kafka.enabled`	Integração Kafka	`false`
`rangerIntegrations.nifi.enabled`	Integração NiFi	`false`
`rangerIntegrations.trino.enabled`	Integração Trino	`false`
`tdp-ranger.solr.persistence.enabled`	Persistência Solr	`true`
`tdp-ranger.solr.persistence.size`	Tamanho do PVC Solr	`1Gi`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Valor padrão
`tdp-ranger.solr.persistence.storageClassName`	StorageClass Solr	`""`
`tdp-ranger.openshift.enabled`	Cria RoleBinding para SCC `anyuid` no OpenShift	`false`
`TDP-Settings.gateway.ingress.enabled`	Habilita Ingress para o Ranger Admin	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Habilita HTTPRoute via Gateway API	`false`

A senha do Ranger Admin deve conter no mínimo 8 caracteres, com letras maiúsculas, minúsculas, número e caractere especial.

Para a lista completa de parâmetros, consulte os valores da versão instalada do chart.

Detalhes de configuração

Banco de dados e auditoria

O Ranger usa PostgreSQL para armazenar políticas, usuários e configurações.
O banco pode ser fornecido pelo próprio release ou por uma instância externa já existente.

Quando usar banco externo, o job de bootstrap cria o banco e o usuário do Ranger e grava as credenciais no Secret esperado pelo deployment.
Esse job só é criado quando o PostgreSQL embutido está desabilitado e `TDP-Settings.externalDatabase.enabled=true`.

O Secret de administrador do PostgreSQL tem o nome definido por `TDP-Settings.externalDatabase.externalSecret.releaseName` (padrão: `tdp-postgresql`) e a chave esperada é `postgres-password`.
Erros de permissão, host incorreto ou Secret ausente aparecem como falha no job de bootstrap.

© Copyright Tecnisys 2026. All rights reserved.



O Solr é usado para auditoria.
Mantenha persistência habilitada e ajuste `tdp-ranger.solr.persistence.size` e `tdp-ranger.solr.persistence.storageClassName` de acordo com retenção, volume de eventos e política operacional do cluster.

Integrações

Quando a automação de integrações está habilitada, o chart executa um job de configuração após a instalação ou atualização.
Esse job atua no lado do Ranger Admin: aguarda o Ranger ficar acessível, cadastra os serviços habilitados em `rangerIntegrations` e cria as políticas declaradas em `defaultPolicies`, quando configuradas.
Ele não configura, por si só, o serviço integrado para consultar o Ranger em tempo de execução.
A configuração do componente integrado deve seguir a documentação específica desse componente.

Consulte Integrações - Ranger para exemplos.

Acesso & Segurança

Segurança

Substitua credenciais de exemplo, mantenha senhas em Secrets e limite permissões do usuário administrador.
Quando usar LDAP/AD, configure bind DN, filtros de usuário/grupo, UserSync, roles e LDAPS conforme o diretório corporativo.

Consulte Segurança - Ranger para detalhes de LDAP/AD, UserSync, LDAPS, credenciais e boas práticas.

Acesso

Liste os Services criados pelo release:

```
kubectl -n <NAMESPACE> get svc -l app.kubernetes.io/instance=<RELEASE_NAME>
```

O Ranger Admin normalmente responde no Service `<RELEASE_NAME>-ranger` pela porta `6080`.

© Copyright Tecnisys 2026. All rights reserved.

Para depuração local:

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME>-ranger 6080:6080
```

Acesso local: `http://localhost:6080`.

Exposição externa

Ingress e Gateway API são mutuamente exclusivos, controlados por `TDP-Settings.gateway.*`. O Ranger Admin é exposto na porta `6080`.

Para os exemplos completos de values (Ingress e Gateway API), consulte Exposição externa.

Solução de problemas

Solução de problemas

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

Verifique primeiro se o Ranger Admin está saudável e se o banco de dados responde. Depois valide jobs de bootstrap, jobs de integração, PVC do Solr, Services e eventos do namespace.

Desinstalação

Desinstalação

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Ranger – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 2.7.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral

O chart `tdp-ranger` concentra a segurança em quatro pontos: credenciais administrativas, credenciais de banco, autenticação LDAP/AD e sincronização de usuários/grupos.
Esses pontos afetam diretamente quem consegue acessar o Ranger Admin, como as políticas são armazenadas e quais identidades aparecem para atribuição de permissões.

Use Secrets para valores sensíveis e mantenha apenas placeholders em arquivos versionados.
Senhas de exemplo devem ser substituídas antes de qualquer ambiente com dados reais.

Modos de autenticação

Modo	Uso típico	Observação
Usuário local `admin`	Instalação inicial e administração emergencial	Troque a senha padrão de exemplo
LDAP/AD	Login com diretório corporativo	Exige bind DN, senha de bind e filtros de busca
UserSync	Sincronização de usuários e grupos para o Ranger	Recomendado junto com LDAP/AD

© Copyright Tecnisys 2026. All rights reserved.

Modo	Uso típico	Observação
LDAPS	Conexão LDAP protegida por TLS	Exige certificado/truststore compatível

O usuário local `admin` é útil para bootstrap e recuperação.
Em ambientes compartilhados, prefira autenticação corporativa e atribuição de roles por grupos.

Credenciais administrativas

Configure `tdp-ranger.ranger.adminUser` e `tdp-ranger.ranger.adminPassword` antes da instalação.
A senha do Ranger Admin deve conter no mínimo 8 caracteres, com letras maiúsculas, minúsculas, número e caractere especial.

Evite registrar a senha real em arquivos versionados.
Quando o ambiente usar ferramenta externa de gestão de segredos, mantenha o arquivo de valores apenas com placeholders e injete o valor no processo de deploy.

Credenciais do banco de dados

O Ranger usa PostgreSQL para armazenar políticas, usuários e configurações.
Quando o banco é externo, o chart executa um job de bootstrap (`db-create-job`) que cria o banco e o usuário do Ranger.

O job só é criado quando as duas condições forem atendidas:

- `tdp-ranger.postgres.enabled=false` — banco embutido desabilitado
- `TDP-Settings.externalDatabase.enabled=true` — PostgreSQL externo habilitado

O job lê as credenciais do administrador PostgreSQL em um Secret cujo nome corresponde ao `releaseName` do release `tdp-postgresql` (padrão: `tdp-postgresql`) e cuja chave esperada é `postgres-password`. Esse Secret é gerado automaticamente pelo chart `tdp-postgresql`.

O Secret de credenciais do Ranger para o banco é esperado em `tdp-ranger.ranger.existingSecret`, com a senha na chave definida por `tdp-`

© Copyright Tecnisys 2026. All rights reserved.

`ranger.ranger.existingSecretPasswordKey`.

Para desabilitar explicitamente o job:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  --set TDP-Settings.externalDatabase.enabled=false
```

LDAP/AD

Habilite LDAP/AD quando o Ranger Admin precisar autenticar usuários em um diretório corporativo.
A configuração deve informar a URL do servidor, o DN de bind, a senha de bind, o DN base, os filtros de busca e os atributos usados para usuários e grupos.

O fluxo recomendado é manter `tdp-ranger.ldap.enabled=false` na configuração base, criar os Secrets necessários e aplicar a configuração LDAP/AD somente no install ou upgrade do ambiente que usará diretório corporativo.
Isso evita habilitar autenticação externa sem bind DN, senha de bind, filtros e certificados já definidos.

Exemplo de aplicação com um arquivo de valores dedicado para LDAP/AD:

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger -f <BASE_VALUES_FILE> -f
<LDAP_VALUES_FILE> -n <NAMESPACE>
```

Crie o Secret da senha de bind no mesmo namespace do Ranger:

```
kubectl create secret generic tdp-ranger-ldap-secret --from-literal=bind-
password='<LDAP_BIND_PASSWORD>' -n <NAMESPACE>
```

Exemplo de configuração:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-ranger:
  ldap:
    enabled: true
    connection:
      url: "ldap://<LDAP_HOST>:389"
      allowInsecure: true
    bind:
      dn: "cn=<LDAP_BIND_USER>,dc=<LDAP_DOMAIN>,dc=com"
      passwordSecret:
        name: "tdp-ranger-ldap-secret"
        key: "bind-password"
    baseDn: "dc=<LDAP_DOMAIN>,dc=com"
    user:
      searchBase: "ou=users,dc=<LDAP_DOMAIN>,dc=com"
      searchFilter: "(&(objectClass=person)(uid={0}))"
      nameAttribute: "uid"
      emailAttribute: "mail"
    group:
      searchBase: "ou=groups,dc=<LDAP_DOMAIN>,dc=com"
      searchFilter: "(objectClass=groupOfNames)"
      memberAttribute: "member"
```

Para Active Directory, ajuste filtros e atributos para o schema usado pelo domínio, como `sAMAccountName` quando aplicável.
Para OpenLDAP ou FreeIPA, valide `uid`, `cn`, `memberOf` e classes de objeto usadas no diretório.

LDAPS e truststore

Use LDAPS quando o diretório exigir conexão protegida por TLS.
Nesse modo, o Ranger precisa confiar no certificado apresentado pelo servidor LDAP.

Crie o Secret com o certificado ou truststore esperado:

```
kubectl create secret generic tdp-ranger-ldap-certs --from-file=truststore.pem=
<LOCAL_PATH> -n <NAMESPACE>
```

Para habilitar o LDAPS, substitua o bloco `connection` pela URL `ldaps://` (porta 636) com `allowInsecure: false` e adicione a seção `tls` referenciando o Secret `tdp-ranger-ldap-certs`:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-ranger:
  ldap:
    connection:
      url: "ldaps://<LDAP_HOST>:636"
      allowInsecure: false
    tls:
      enabled: true
      truststore:
        secretName: "tdp-ranger-ldap-certs"
        secretKey: "truststore.pem"
        path: "/etc/ranger/ldap-certs"
```

Se a cadeia de certificados estiver incompleta, o login LDAP pode falhar mesmo com usuário e senha corretos.
Valide CA, SAN do certificado e resolução DNS do host LDAP.

UserSync

O UserSync sincroniza usuários e grupos do diretório com o Ranger Admin.
Isso permite criar políticas usando identidades corporativas sem cadastrar usuários manualmente.

Quando LDAP/AD estiver habilitado, configure `tdp-ranger.usersync.enabled=true` e ajuste fonte de sincronização, intervalo, filtros e recursos.
O UserSync usa um initContainer para aguardar o Ranger Admin ficar disponível antes de iniciar a sincronização.
No fluxo dedicado de LDAP/AD, o UserSync usa a imagem `registry.tecnisys.com.br/tdp/images/ranger-usersync:2.7.0` e carrega `install.properties` a partir de ConfigMap, aplicando os segredos em tempo de execução. Personalize recursos via `tdp-ranger.usersync.resources` e ajustes finos de LDAP via `tdp-ranger.usersync.ldap`.

Exemplo:

```
tdp-ranger:
  usersync:
    enabled: true
    syncSource: "ldap"
    syncInterval: 60
```

© Copyright Tecnisys 2026. All rights reserved.

Atribuição de roles

Use `tdp-ranger.usersync.roleAssignment.rules` para atribuir roles do Ranger a usuários ou grupos sincronizados.
Esse recurso evita que a administração dependa exclusivamente do usuário local `admin`.

```
tdp-ranger:
  usersync:
    roleAssignment:
      enabled: true
      rules:
        - role: "ROLE_SYS_ADMIN"
          users: []
          groups:
            - "cn=ranger_admins,ou=groups,dc=<LDAP_DOMAIN>,dc=com"
        - role: "ROLE_ADMIN_AUDITOR"
          users: []
          groups:
            - "cn=ranger_auditors,ou=groups,dc=<LDAP_DOMAIN>,dc=com"
```

Roles suportadas incluem `ROLE_SYS_ADMIN`, `ROLE_KEY_ADMIN`, `ROLE_ADMIN_AUDITOR`, `ROLE_KEY_ADMIN_AUDITOR` e `ROLE_USER`.
Use grupos do diretório para reduzir alterações manuais no Ranger Admin.

Parâmetros principais

Parâmetro	Descrição
`tdp-ranger.ranger.adminUser`	Usuário administrador local do Ranger
`tdp-ranger.ranger.adminPassword`	Senha do administrador local
`tdp-ranger.ranger.dbPassword`	Senha do usuário de banco do Ranger
`tdp-ranger.ranger.existingSecret`	Secret com credenciais do banco do Ranger
`tdp-ranger.ldap.enabled`	Habilita autenticação LDAP/AD

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição
`tdp-ranger.ldap.connection.url`	URL do servidor LDAP ou LDAPS
`tdp-ranger.ldap.connection.allowInsecure`	Permite conexão LDAP sem TLS
`tdp-ranger.ldap.bind.passwordSecret`	Secret com senha de bind
`tdp-ranger.ldap.user.searchFilter`	Filtro de busca de usuários
`tdp-ranger.ldap.group.searchFilter`	Filtro de busca de grupos
`tdp-ranger.ldap.tls.enabled`	Habilita TLS/LDAPS
`tdp-ranger.usersync.enabled`	Habilita sincronização de usuários e grupos
`tdp-ranger.usersync.roleAssignment.enabled`	Habilita atribuição automática de roles

Solução de problemas

Sintoma	Possível causa	Verificação
Login local falha	Senha admin incorreta ou Secret desatualizado	Verifique Secret e eventos do pod
Login LDAP falha	Bind DN, senha ou filtro incorreto	Teste DN, filtro e Secret de bind
LDAPS falha	Certificado não confiável ou host não bate com certificado	Verifique CA, SAN e `truststore.pem`

© Copyright Tecnisys 2026. All rights reserved.

Sintoma	Possível causa	Verificação
Usuários não aparecem	UserSync desabilitado ou filtros restritivos	Verifique logs do UserSync
Roles não são atribuídas	DN de grupo não bate com o diretório	Compare o DN real do grupo com a regra

Comandos úteis:

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get secrets
kubectl -n <NAMESPACE> logs deploy/<RELEASE_NAME>-ranger
```

Boas práticas

- Troque todas as senhas de exemplo antes de ambientes com dados reais.
- Use Secrets ou ferramenta externa para valores sensíveis.
- Prefira LDAPS para diretórios corporativos.
- Use grupos LDAP/AD para administrar roles e políticas.
- Mantenha o usuário local `admin` apenas para bootstrap e recuperação controlada.
- Valide UserSync antes de criar políticas dependentes de grupos.

© Copyright Tecnisys 2026. All rights reserved.

Ranger – Integração

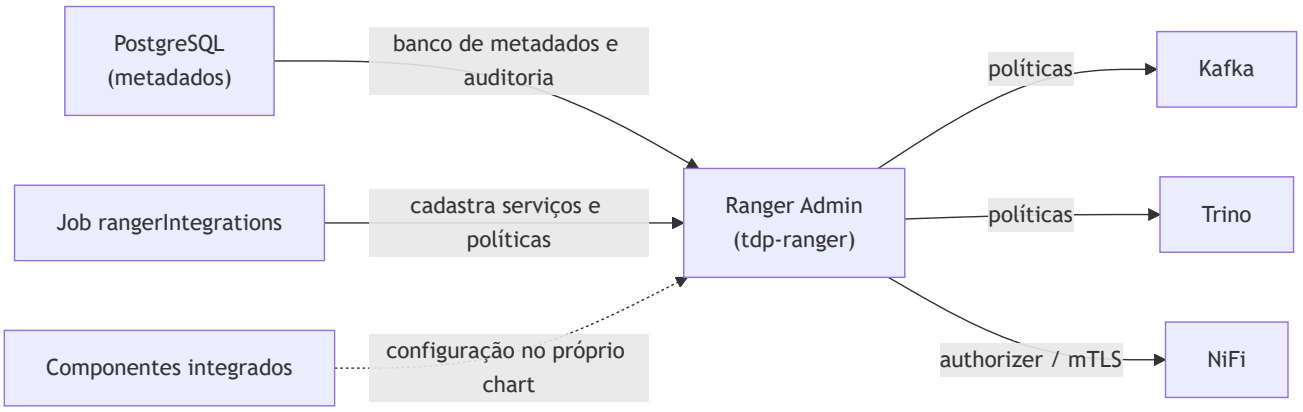
CHART

Version 3.0.1 | Type application | AppVersion 2.7.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral das integrações



O chart `tdp-ranger` pode usar um PostgreSQL externo (`tdp-postgresql`) como banco de metadados e de auditoria — consulte Configuração do Ranger.

O bloco `rangerIntegrations` automatiza uma parte específica do fluxo de integração: o cadastro de serviços e políticas no Ranger Admin.
Quando `rangerIntegrations.configJob.enabled=true`, o chart executa um job pós-instalação ou pós-atualização que aguarda o Ranger Admin responder, lê as integrações habilitadas, cadastra ou atualiza os serviços correspondentes no Ranger Admin e cria as políticas declaradas em `defaultPolicies`, quando configuradas.
Sem esse job, esse cadastro precisa ser feito manualmente na interface ou API do Ranger Admin.

> **ESCOPO DA AUTOMAÇÃO**
>
> Essa automação atua no lado do Ranger Admin.
> A configuração do serviço integrado para consultar o Ranger, quando necessária, deve ser feita conforme a documentação específica do respectivo

© Copyright Tecnisys 2026. All rights reserved.

componente.
Habilitar `rangerIntegrations.<SERVICE_KEY>.enabled=true` não configura o componente de ponta a ponta.

AUTOMAÇÃO DESABILITADA POR PADRÃO

`rangerIntegrations.configJob.enabled` é `false` por padrão.
Para automatizar o cadastro no Ranger Admin, habilite esse parâmetro no seu arquivo de valores.

Padrão de configuração

O mesmo modelo de configuração pode aparecer em três escopos, dependendo de como o chart é usado no ambiente:

Escopo	Uso
`rangerIntegrations.*`	Consumido pelos recursos renderizados pelo `tdp-ranger`, incluindo job e Config-Map de integração
`tdp-ranger.rangerIntegrations.*`	Repassado ao subchart do Apache Ranger
`global.rangerIntegrations.*`	Compartilhado com subcharts que consomem valores globais

Use o escopo que corresponde ao pacote implantado.
Quando o cadastro no Ranger Admin depender do job automático, garanta que `rangerIntegrations.configJob.enabled=true`.

Pré-requisitos

- Ranger Admin implantado e acessível pelo Service interno.
- Senha do usuário administrador configurada e disponível para o job de integrações.
- Serviço de destino implantado e acessível por rede.
- Nome do serviço no Ranger definido em `serviceName`.

© Copyright Tecnisys 2026. All rights reserved.



- Credenciais e endpoints compatíveis com o tipo de serviço cadastrado no Ranger Admin.
- Configuração do componente integrado feita no próprio chart ou documentação do componente, quando ele precisar consultar o Ranger em tempo de execução.
- Políticas declaradas em `defaultPolicies` revisadas antes de aplicar em ambiente compartilhado.

Job de integrações

O job de integrações é executado após instalação ou atualização do release.
Ele aguarda o Ranger Admin responder, instala dependências Python no container do job e executa o script que cadastra serviços e políticas no Ranger Admin.

Parâmetros principais:

Parâmetro	Descrição
`rangerIntegrations.configJob.enabled`	Habilita o job automático
`rangerIntegrations.configJob.image`	Imagem usada pelo container de configuração
`rangerIntegrations.configJob.imagePullPolicy`	Política de pull da imagem
`rangerIntegrations.configJob.rangerReadyTimeout`	Tempo máximo de espera pelo Ranger Admin
`rangerIntegrations.configJob.retryInterval`	Intervalo entre tentativas

Exemplo:

```
rangerIntegrations:
  configJob:
    enabled: true
```

© Copyright Tecnisys 2026. All rights reserved.

```
    rangerReadyTimeout: 600
    retryInterval: 10
```

Exemplos de integrações

As seções seguintes mostram exemplos de configuração aceitos por `rangerIntegrations`.
Elas não devem ser lidas como uma lista conceitual fixa de tudo que o Apache Ranger pode proteger.

Kafka

A configuração de Kafka registra um serviço do tipo Kafka no Ranger Admin.
Ela precisa dos endpoints de bootstrap, conexão com ZooKeeper quando exigida pelo plugin e protocolo de segurança.
Essa configuração demonstra o cadastro no Ranger Admin e a criação das políticas declaradas em `defaultPolicies`, quando existirem.
Ela não afirma que o chart do Kafka configure automaticamente o plugin do Ranger no broker.

```
rangerIntegrations:
  configJob:
    enabled: true
  kafka:
    enabled: true
    serviceName: "tdp_kafka"
    serviceDisplayName: "TDP Kafka Cluster"
    connection:
      bootstrapServers: "<KAFKA_BOOTSTRAP_SERVICE>.
<NAMESPACE>.svc.cluster.local:9092"
      zookeeperConnect: "<ZOOKEEPER_SERVICE>.<NAMESPACE>.svc.cluster.local:2181"
      securityProtocol: "PLAINTEXT"
    credentials:
      username: "<USERNAME>"
      password: "<PASSWORD>"
    defaultPolicies: []
```

Use `defaultPolicies` para criar políticas declaradas de tópicos, cluster, transactional ID e delegation token quando isso fizer parte do bootstrap do ambiente.
Revise grupos e usuários antes de aplicar políticas amplas como acesso total.

© Copyright Tecnisys 2026. All rights reserved.

Para habilitação rápida via linha de comando:

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  --set rangerIntegrations.kafka.enabled=true \
  --set rangerIntegrations.kafka.connection.bootstrapServers=
<KAFKA_BOOTSTRAP_SERVERS> \
  --set rangerIntegrations.kafka.connection.zookeeperConnect=<ZOOKEEPER_CONNECT>
```

Trino

A configuração de Trino registra no Ranger Admin um serviço com URL JDBC e driver do Trino.
Ela permite declarar políticas para catálogos, schemas, tabelas e colunas no Ranger Admin.
Esse cadastro não substitui a configuração do controle de acesso Ranger no chart do Trino; o Trino também precisa estar configurado para consultar o Ranger em tempo de execução.

```
rangerIntegrations:
  configJob:
    enabled: true
  trino:
    enabled: true
    serviceName: "tdp_trino"
    serviceDisplayName: "TDP Trino Cluster"
    connection:
      jdbcUrl: "jdbc:trino://<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local:8080"
      jdbcDriverClassName: "io.trino.jdbc.TrinoDriver"
    credentials:
      username: "<USERNAME>"
      password: "<PASSWORD>"
    defaultPolicies: []
```

As credenciais devem representar um usuário técnico capaz de validar a conexão do serviço cadastrado no Ranger.
As políticas devem refletir o modelo de dados do ambiente, evitando liberar `catalog`, `schema` e `table` como `*` sem revisão.

© Copyright Tecnisys 2026. All rights reserved.

NiFi genérico

A configuração NiFi genérica registra um serviço NiFi no Ranger Admin.
Ela pode ser útil para cenários simples, mas não cobre todo o fluxo de produção em que o NiFi usa Ranger como authorizer principal.
Essa configuração demonstra o cadastro no Ranger Admin e a criação das políticas declaradas em `defaultPolicies`, quando existirem.
Ela não afirma que o chart do NiFi configure automaticamente o authorizer do Ranger.

```
rangerIntegrations:
  configJob:
    enabled: true
  nifi:
    enabled: true
    serviceName: "tdp_nifi"
    serviceDisplayName: "TDP NiFi Cluster"
    connection:
      nifiUrl: "http://<NIFI_SERVICE>.<NAMESPACE>.svc.cluster.local:8080"
      internalUrl: "http://<NIFI_HEADLESS_SERVICE>.
<NAMESPACE>.svc.cluster.local:8080"
      authenticationType: "none"
    credentials:
      username: "<USERNAME>"
      password: "<PASSWORD>"
    defaultPolicies: []
```

Use esse modo apenas quando ele corresponder ao modelo de segurança do NiFi implantado.
Para NiFi com autorização centralizada no Ranger, use o padrão com mTLS descrito abaixo.

> **ARQUIVO DE OVERLAY DEDICADO**
>
> Para integração do NiFi com o Ranger como authorizer, utilize o arquivo de overlay dedicado fornecido com o chart. Ele já alinha a definição do serviço com a documentação do plugin Apache Ranger para NiFi, apontando `nifi.url` para `/nifi-api/resources` e usando um hostname que esteja presente nos SANs dos certificados do NiFi.

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger \
  -n <NAMESPACE> \
  -f values-nifi-ranger-authorizer.yaml
```

NiFi como authorizer via Ranger

Quando o NiFi usa Ranger como authorizer principal, o cadastro do serviço no Ranger Admin precisa ser compatível com o fluxo de mTLS entre Ranger e NiFi.
Nesse cenário, o Ranger acessa o endpoint `/nifi-api/resources`, cria usuários técnicos e cria as políticas declaradas em `defaultPolicies` para que o cluster NiFi consiga operar.
Esse fluxo exige também a configuração correspondente no chart do NiFi para que o NiFi consulte o Ranger como authorizer.

Antes da instalação, crie o Secret com a senha estável do JKS:

```
kubectl -n <NAMESPACE> create secret generic ranger-nifi-client-tls-password --
from-literal=password='<KEYSTORE_PASSWORD>' --dry-run=client -o yaml | kubectl
apply -f -
```

Para uma primeira instalação, também é possível criar o Secret em um comando direto:

```
kubectl -n <NAMESPACE> create secret generic ranger-nifi-client-tls-password --
from-literal=password='<KEYSTORE_PASSWORD>'
```

Não rotacione essa senha casualmente.
Se ela mudar sem reemitir o Secret de certificado cliente, o Ranger pode falhar ao abrir `keystore.jks` e `truststore.jks`.

Configuração principal:

```
rangerIntegrations:
  configJob:
```

© Copyright Tecnisys 2026. All rights reserved.



```
    enabled: true
  nifi:
    enabled: true
    serviceName: "tdp_nifi"
    serviceDisplayName: "TDP NiFi Cluster"
    technicalUsers:
      - "CN=ranger-nifi-client"
      - "CN=nifi-controller"
    tlsClientCertificate:
      autoGenerate:
        enabled: true
        secretName: "ranger-nifi-client-tls"
        passwordSecretName: "ranger-nifi-client-tls-password"
        passwordSecretKey: "password"
        issuerRef:
          name: "nifi-issuer"
          kind: "Issuer"
          group: "cert-manager.io"
        commonName: "ranger-nifi-client"
        dnsNames:
          - "ranger-nifi-client"
    connection:
      nifiUrl: "https://nifi-headless.<NAMESPACE>.svc.cluster.local:8443/nifi-
api/resources"
      internalUrl: "https://nifi-headless.
<NAMESPACE>.svc.cluster.local:8443/nifi-api/resources"
      authenticationType: "SSL"
      sslUseDefaultContext: "false"
      ssl:
        keystore: "/opt/ranger/certs/keystore.jks"
        keystoreType: "jks"
        keystorePasswordSecret:
          name: "ranger-nifi-client-tls-password"
          key: "password"
        truststore: "/opt/ranger/certs/truststore.jks"
        truststoreType: "jks"
        truststorePasswordSecret:
          name: "ranger-nifi-client-tls-password"
          key: "password"
      additionalConfigs:
        policy.download.auth.users: "nifi"
```

O certificado cliente é criado por cert-manager quando `tlsClientCertificate.autoGenerate.enabled=true`.

© Copyright Tecnisys 2026. All rights reserved.



O Common Name configurado para o certificado deve bater com a identidade de administrador Ranger configurada no NiFi.

O chart também monta o material TLS em `/opt/ranger/certs` para que o Ranger use `keystore.jks` e `truststore.jks`.
Use o Service headless interno do NiFi quando os certificados dos nós incluírem esse hostname nos SANs.

Políticas declaradas

Cada integração aceita `defaultPolicies`.
Esse bloco permite criar políticas no Ranger Admin durante o bootstrap, quando configurado.

No caso de NiFi como authorizer, as políticas técnicas declaradas para o bootstrap devem permitir:

- `CN=ranger-nifi-client` em `/resources`.
- `CN=nifi-controller` em `/controller` e `/system`.
- Identidades dos nós NiFi em `/proxy`.

Essas políticas são técnicas e não substituem políticas humanas de acesso ao fluxo. Crie políticas de usuários e grupos reais depois que LDAP/AD e UserSync estiverem validados.

Validação

Verifique se o job foi criado e concluiu com sucesso:

```
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<RELEASE_NAME>-ranger-integrations
```

No Ranger Admin, valide:

- serviço criado com o `serviceName` esperado;
- conexão do serviço sem erro;
- políticas declaradas em `defaultPolicies` aplicadas, quando configuradas;
- usuários técnicos criados quando a integração NiFi com mTLS estiver habilitada.

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Sintoma	Possível causa	Verificação
Job não é criado	`rangerIntegrations.configJob.enabled=false`	Verifique o arquivo de valores aplicado
Job falha aguardando Ranger	Ranger Admin indisponível ou senha admin incorreta	Verifique pods, Service e Secret admin
Serviço não aparece no Ranger	Integração específica desabilitada	Verifique `rangerIntegrations.<SERVICE_KEY>.enabled`
NiFi mTLS falha	Senha JKS ausente ou certificado não emitido	Verifique Secrets e Certificate do cert-manager
Políticas não aplicam	Usuários/grupos não existem ou DN incorreto	Valide UserSync e nomes usados nas políticas

Depois de corrigir a causa, execute novo `helm upgrade` para recriar o hook de integração.

© Copyright Tecnisys 2026. All rights reserved.

Spark

CHART

Version 3.0.1 | Type application | AppVersion 4.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

S3 suportado | Ingress suportado | Gateway API suportado

O que é o Apache Spark no TDP?

O Apache Spark é o motor de processamento distribuído usado no TDP para transformações de dados, pipelines ETL e cargas batch.
Enquanto o Trino é voltado a consultas SQL interativas, o Spark é mais adequado para processamentos longos, transformações complexas e gravação de resultados em armazenamento distribuído.

No TDP Kubernetes, o Spark normalmente é consumido por clientes como Airflow e Jupyter.
O Airflow agenda e dispara jobs, enquanto o Jupyter permite uso interativo em notebooks.

> **PARA SABER MAIS**
>
> Consulte Apache Spark - Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

O chart `tdp-spark` é um wrapper sobre o chart upstream do Spark. Os valores se distribuem em dois blocos:

© Copyright Tecnisys 2026. All rights reserved.

- `spark:` — valores do chart upstream: executor, driver, workers, imagens, `sparkConf`, `sparkContext`.
- `tdp-spark:` — valores do wrapper TDP: `hadoopConfig`, `customSparkConfig.properties`, integrações (Jupyter, Airflow, Delta Lake, Iceberg), ServiceAccount.

```
spark:
  worker:
    replicaCount: 2

tdp-spark:
  hadoopConfig:
    "fs.defaultFS": "s3a://warehouse"
```

Quando estiver em dúvida sobre qual bloco usar, consulte a seção Parâmetros principais desta página ou `helm show values`.

Arquitetura master-worker

O chart `tdp-spark` empacota e configura um chart upstream do Spark, adicionando configurações e integrações específicas da plataforma TDP.
A implantação segue a arquitetura standalone com master e workers:

Componente	Função
Master	Gerencia o cluster Spark, recebe aplicações e distribui trabalho para os workers
Workers	Executam os executors das aplicações Spark
History Server	Disponibiliza a interface de histórico das execuções quando habilitado

© Copyright Tecnisys 2026. All rights reserved.

Componente	Função
ConfigMaps TDP	Renderizam `core-site.xml`, `spark-defaults.conf` e configurações de integração quando habilitados

O ponto de submissão do cluster é o master Spark, normalmente acessado por clientes no formato `spark://<SPARK_MASTER_SERVICE>:7077`.
Drivers externos, como Airflow ou notebooks Jupyter, precisam alcançar esse endpoint para submeter jobs.

Visão geral

Propriedade	Valor
Chart	`tdp-spark`
Versão do Spark	4.0.2
Versão do Chart	3.0.1
AppVersion do chart	4.0.2
Registry	`oci://registry.tecnisys.com.br/tdp/charts/tdp-spark`

Páginas relacionadas

- Integrações - Spark: Jupyter, Airflow, Delta Lake, Iceberg e configurações S3/S3A usadas por integrações.
- Segurança - Spark: credenciais S3/S3A, placeholders, Secrets e boas práticas.
- Exposição externa - Spark: Ingress (com reverse proxy obrigatório) e Gateway API para a UI do Spark.

Pré-requisitos

Antes de instalar ou customizar o chart `tdp-spark`, confirme:

© Copyright Tecnisys 2026. All rights reserved.



- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível para volumes persistentes.
- Registry OCI da Tecnisys acessível pelo ambiente de instalação.
- Quando usar S3/S3A: endpoint, credenciais, bucket e permissões disponíveis para os pods Spark.
- Quando usar Jupyter ou Airflow: conectividade desses clientes até o serviço master do Spark.

Instalação

Instalação

Exemplo

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-spark
-n <NAMESPACE> --create-namespace
```

Ajuste recursos, réplicas e configurações Spark pelo seu arquivo de valores ou por `--set`, conforme a necessidade do ambiente.
O chart é um wrapper: parte dos parâmetros pertence ao bloco `spark` do chart upstream e parte pertence ao bloco `tdp-spark` do wrapper TDP.

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Observação
`spark.enabled`	Habilita a implantação do Spark	Controla a dependência upstream
`spark.image.repository`	Repositório da imagem Spark	Validar o valor efetivo no `values.yaml` da versão instalada

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Observação
`spark.image.tag`	Tag da imagem Spark	Alinhada à versão Spark empacotada pelo chart
`spark.master.resources`	Recursos do master	Requests/limits do processo master
`spark.worker.replicaCount`	Quantidade de workers	Use `2` como referência inicial
`spark.worker.autoscaling.enabled`	Habilita HPA dos workers	Use `false` como referência inicial
`spark.historyServer.enabled`	Habilita History Server	Use `true` como referência inicial
`spark.sparkConf`	Mapa de propriedades Spark	Inclui propriedades `spark.*` e `spark.hadoop.*`
`tdp-spark.hadoopConfig`	Valores de `core-site.xml` renderizados pelo wrapper	Usado para propriedades Hadoop/S3A
`tdp-spark.customSparkConfig.properties`	Conteúdo de `spark-defaults.conf` renderizado pelo wrapper	Usado para defaults adicionais do Spark
`tdp-spark.integration.jupyter.enabled`	Habilita configuração de integração com Jupyter	Renderiza ConfigMap quando habilitado
`tdp-spark.integration.airflow.enabled`	Habilita bloco de configuração para Airflow	Guarda defaults de cliente Spark
`tdp-spark.deltaLake.enabled`	Bloco opcional Delta Lake	Não altera `spark.sparkConf` automaticamente

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Observação
`tdp-spark.iceberg.enabled`	Bloco opcional Iceberg	Não altera `spark.sparkConf` automaticamente

O chart usa um bloco `tdp-spark.serviceAccount` para templates do wrapper TDP e um bloco `spark.serviceAccount` para a dependência upstream.
Mantenha ambos alinhados quando precisar customizar ServiceAccount.

Detalhes de configuração

Configuração Spark

Use `spark.sparkConf` para propriedades que o próprio Spark deve receber, como executor, driver, event log e chaves `spark.hadoop.*`.
Use `tdp-spark.hadoopConfig` quando precisar renderizar `core-site.xml` pelo wrapper TDP.
Use `tdp-spark.customSparkConfig.properties` quando precisar renderizar conteúdo adicional de `spark-defaults.conf`.

A separação é importante: `core-site.xml` é lido pelo Hadoop antes de muitas propriedades do Spark, enquanto `spark-defaults.conf` define defaults de submissão e execução.
Para detalhes de credenciais e S3/S3A, consulte Segurança - Spark.

Armazenamento S3/S3A

S3/S3A é um ponto central de configuração do chart.
As propriedades podem ser configuradas por `spark.sparkConf`, por `tdp-spark.hadoopConfig` ou por `tdp-spark.customSparkConfig.properties`.

Na página geral, mantenha apenas a visão de alto nível: o Spark precisa de endpoint, credenciais e permissões para acessar o armazenamento.
Os exemplos completos e as boas práticas de credenciais ficam em Segurança - Spark.

Integrações e formatos de tabela

© Copyright Tecnisys 2026. All rights reserved.



O chart possui blocos para integração com Jupyter e Airflow, além de blocos opcionais para Delta Lake e Iceberg.
Esses blocos organizam valores de configuração, mas Delta Lake e Iceberg não modificam automaticamente `spark.sparkConf`.
As propriedades Spark e dependências necessárias devem ser fornecidas em `spark.sparkConf` e `tdp-spark.customSparkConfig.properties`.

Consulte Integrações - Spark para exemplos e validações por integração.

Recursos, workers e autoscaling

O master coordena o cluster; os workers executam os executors.
Aumente `spark.worker.replicaCount` quando precisar de mais capacidade fixa de execução.
Use `spark.worker.autoscaling.enabled` apenas quando o cluster tiver suporte para HPA e métricas disponíveis.

> **PRÉ-REQUISITOS PARA HPA**
>
> - O **Metrics Server** deve estar instalado no cluster.
> - Os recursos (`resources.requests.cpu` e `resources.requests.memory`) dos workers **devem ser configurados** para que o HPA funcione.

Exemplo de configuração de HPA para workers:

```
spark:
  worker:
    replicaCount: 2
    autoscaling:
      enabled: true
      minReplicas: 2
      maxReplicas: 10
      targetCPU: 70
      targetMemory: 80
    resources:
      requests:
        cpu: "1"
        memory: "2Gi"
      limits:
```

© Copyright Tecnisys 2026. All rights reserved.

```
        cpu: "2"
        memory: "4Gi"
```

Ajuste recursos de master, workers, driver e executors em conjunto.
Configurar muitos executors sem recursos suficientes nos workers pode gerar fila, falhas de agendamento ou pressão de memória.

Acesso & Segurança

Exposição externa

O chart suporta exposição externa via **Ingress** (suportado pelo subchart upstream Bitnami, configurado em `spark.ingress.*`) e via **Gateway API** (configurado em `gatewayApi.spark.*`).

> **REVERSE PROXY OBRIGATÓRIO PARA INGRESS**
>
> Para que a UI do master Spark e os links de workers e aplicações funcionem corretamente através de um Ingress, o **Spark Reverse Proxy nativo deve ser habilitado**:
>
> ```
> spark:
>   master:
>     configOptions: "-Dspark.ui.reverseProxy=true -
> Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
>   worker:
>     configOptions: "-Dspark.ui.reverseProxy=true -
> Dspark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME>"
>   sparkConf:
>     "spark.ui.reverseProxy": "true"
>     "spark.ui.reverseProxyUrl": "http://<SPARK_HOSTNAME>"
> ```
>
> Sem essa configuração, os links "Back to Master" e os redirecionamentos de workers apontam para endereços internos do cluster, inacessíveis externamente.
>
> Ao submeter aplicações, informe os mesmos parâmetros no `spark-submit`:

© Copyright Tecnisys 2026. All rights reserved.

```
spark-submit \
  --conf spark.ui.reverseProxy=true \
  --conf spark.ui.reverseProxyUrl=http://<SPARK_HOSTNAME> \
  ...
```

Sem esses parâmetros, ao navegar pelas abas da Spark Web UI você é redirecionado para a home em vez das informações do job.

Consulte Exposição Externa — Spark para os exemplos completos de Ingress e Gateway API.

Acesso local

Para depuração ou acesso temporário sem Ingress, use `port-forward` para a UI do master Spark:

Liste os Services criados pelo release:

```
kubectl -n <NAMESPACE> get svc -l app.kubernetes.io/instance=<RELEASE_NAME>
```

Para depuração local, use port-forward para a UI do master Spark:

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME>-master-svc 8080:80
```

Solução de problemas

Solução de problemas

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

Verifique primeiro se o master e os workers estão saudáveis.
Depois valide Services, configurações de S3/S3A e logs dos jobs submetidos pelo cliente usado no ambiente.

© Copyright Tecnisys 2026. All rights reserved.

Desinstalação

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Para a lista completa de parâmetros, consulte os valores da versão instalada do chart.

© Copyright Tecnisys 2026. All rights reserved.

Spark – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 4.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral

O chart `tdp-spark` concentra a segurança principalmente na forma como o Spark acessa armazenamento S3/S3A e como valores sensíveis são fornecidos à configuração. Os valores de S3/S3A devem ser tratados como placeholders em arquivos versionados e fornecidos por Secrets ou ferramentas externas.

A ideia prática é simples: Spark precisa de endpoint, credenciais e permissões para ler e gravar dados; a documentação deve mostrar onde esses valores entram, mas não deve versionar segredos reais.

Modos de acesso a armazenamento

Modo	Onde configurar	Uso típico
`spark.sparkConf`	Chaves `spark.hadoop.fs.s3a.*`	Propriedades diretamente repassadas ao Spark
`tdp-spark.hadoopConfig`	Chaves `fs.s3a.*`	Renderização de `core-site.xml` pelo wrapper TDP
`tdp-spark.customSparkConfig.properties`	Linhas de `spark-defaults.conf`	Defaults adicionais usados por clientes Spark

Use apenas placeholders em arquivos versionados.
Forneça valores reais por arquivo privado de valores, Secret ou mecanismo externo de

© Copyright Tecnisys 2026. All rights reserved.

gestão de segredos.

Pré-requisitos

- Endpoint S3/S3A acessível a partir dos pods Spark.
- Credenciais com permissão adequada no bucket ou prefixo usado pelos jobs.
- Bucket ou warehouse criado antes da execução dos pipelines.
- Configuração de TLS/HTTP coerente com o endpoint usado.
- NetworkPolicy permitindo tráfego dos pods Spark até o serviço de armazenamento.

Credenciais S3/S3A

As credenciais podem ser informadas via `spark.sparkConf` quando o ambiente exigir que o Spark receba as chaves como propriedades `spark.hadoop.*`:

```
spark:
  sparkConf:
    "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>:<S3_PORT>"
    "spark.hadoop.fs.s3a.access.key": "<ACCESS_KEY>"
    "spark.hadoop.fs.s3a.secret.key": "<SECRET_KEY>"
    "spark.hadoop.fs.s3a.path.style.access": "true"
    "spark.hadoop.fs.s3a.connection.ssl.enabled": "false"
```

Não use credenciais reais nesse exemplo.
Substitua os placeholders por valores vindos de mecanismo seguro no processo de instalação.

Configuração do core-site.xml via hadoopConfig

O bloco `tdp-spark.hadoopConfig` é renderizado pelo chart como `core-site.xml`.
Esse arquivo é lido pelo Hadoop antes de muitos defaults do Spark, por isso é útil para propriedades de filesystem e autenticação Hadoop.

Modelo:

```
tdp-spark:
  hadoopConfig:
    "fs.s3a.endpoint": "http://<S3_ENDPOINT>:<S3_PORT>"
    "fs.s3a.access.key": "<ACCESS_KEY>"
```

© Copyright Tecnisys 2026. All rights reserved.

```
    "fs.s3a.secret.key": "<SECRET_KEY>"
    "fs.s3a.path.style.access": "true"
```

O chart também usa esse bloco para configurações como autenticação Hadoop simples, `fs.defaultFS` e parâmetros S3A.
Valide o conjunto final com os valores aplicados no ambiente.

Configuração do spark-defaults.conf via customSparkConfig

O bloco `tdp-spark.customSparkConfig.properties` é renderizado como `spark-defaults.conf`.
Use esse caminho para defaults adicionais do Spark, como event log, Ivy local, propriedades Hadoop e ajustes de S3A.

Modelo:

```
tdp-spark:
  customSparkConfig:
    properties: |
      spark.hadoop.fs.s3a.endpoint http://<S3_ENDPOINT>:<S3_PORT>
      spark.hadoop.fs.s3a.access.key <ACCESS_KEY>
      spark.hadoop.fs.s3a.secret.key <SECRET_KEY>
      spark.hadoop.fs.s3a.path.style.access true
```

Mantenha senhas, access keys e secret keys fora do repositório.

Jupyter e Airflow

As integrações com Jupyter e Airflow podem carregar defaults Spark que apontam para o master e para configurações de execução.
Se esses clientes acessarem S3/S3A diretamente, eles também precisam receber as propriedades e credenciais necessárias de forma segura.

A configuração de cliente não substitui a proteção das credenciais.
Revise os valores aplicados em `tdp-spark.integration.jupyter.sparkConfig` e `tdp-spark.integration.airflow.sparkConfig` antes de expor notebooks ou DAGs a usuários finais.

Parâmetros principais

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Papel	Quando alterar
`spark.sparkConf`	Define propriedades Spark e `spark.hadoop.*`	Quando o runtime Spark precisa receber propriedades diretamente
`tdp-spark.hadoopConfig`	Renderiza `core-site.xml`	Quando a configuração Hadoop/S3A deve ser compartilhada pelos pods
`tdp-spark.customSparkConfig.properties`	Renderiza `spark-defaults.conf`	Quando clientes e jobs precisam de defaults adicionais
`tdp-spark.integration.jupyter.sparkConfig`	Defaults Spark para Jupyter	Quando notebooks submetem jobs ao cluster Spark
`tdp-spark.integration.airflow.sparkConfig`	Defaults Spark para Airflow	Quando DAGs submetem jobs ao cluster Spark

Solução de problemas

Situação	O que verificar
Jobs Spark falham com erro de acesso S3	Endpoint, access key, secret key e permissões no bucket
`Connection refused` ao S3	Nome do serviço, namespace, porta e NetworkPolicy
`AccessDeniedException`	Permissões da credencial no bucket ou prefixo
Configuração S3A não aparece no pod	Renderização de `core-site.xml`, `spark-defaults.conf` e valores aplicados no

© Copyright Tecnisys 2026. All rights reserved.

Situação	O que verificar
	release
Notebook ou DAG não acessa dados	Defaults de Jupyter/Airflow e credenciais disponíveis para o cliente

Boas práticas

- Não versionar `access.key`, `secret.key`, tokens ou endpoints sensíveis em repositório Git.
- Usar arquivos de valores privados ou mecanismos de Secret externos para credenciais reais.
- Separar valores por ambiente, como desenvolvimento, homologação e produção.
- Rotacionar credenciais periodicamente e aplicar `helm upgrade` quando necessário.
- Dar às credenciais apenas as permissões necessárias no bucket ou prefixo usado pelos jobs.

© Copyright Tecnisys 2026. All rights reserved.

Spark – Integração

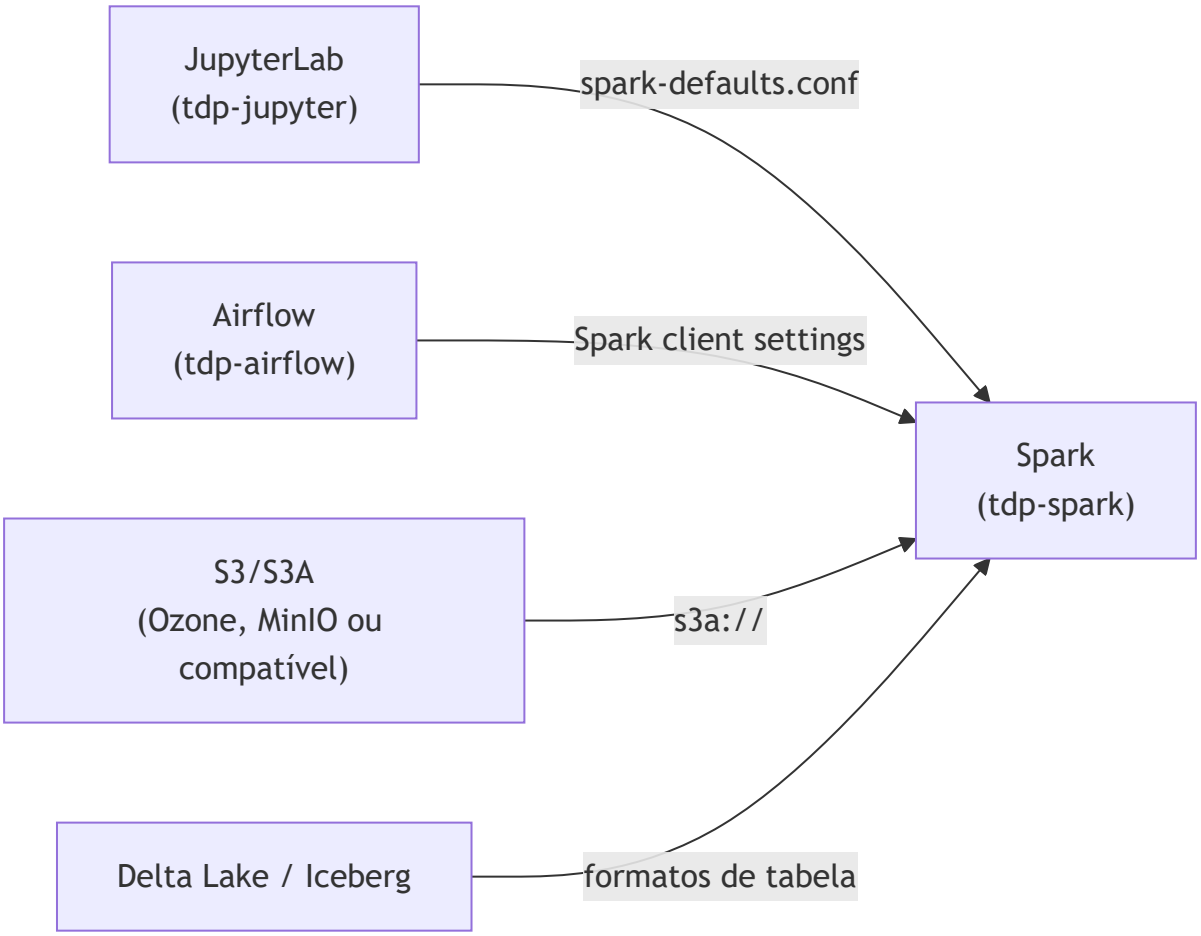
CHART

Version 3.0.1 | Type application | AppVersion 4.0.2

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral



O chart `tdp-spark` organiza integrações por meio de blocos de valores e ConfigMaps. Jupyter e Airflow recebem configurações de cliente Spark, enquanto Delta Lake e Iceberg são blocos opcionais para cenários de formatos de tabela.

© Copyright Tecnisys 2026. All rights reserved.



O ponto mais importante é que esses blocos não substituem a configuração Spark do ambiente.
Quando uma integração precisa de catálogos, bibliotecas, endpoint S3 ou metastore, complete também `spark.sparkConf` e `tdp-spark.customSparkConfig.properties`.

Padrão de configuração

Há três caminhos principais para configurar Spark e armazenamento:

Bloco	Papel
`spark.sparkConf`	Propriedades Spark e `spark.hadoop.*` repassadas ao runtime Spark
`tdp-spark.hadoopConfig`	Valores renderizados como `core-site.xml` pelo wrapper TDP
`tdp-spark.customSparkConfig.properties`	Conteúdo renderizado como `spark-defaults.conf` pelo wrapper TDP

Use `spark.sparkConf` para propriedades do Spark, `hadoopConfig` para configurações Hadoop/S3A compartilhadas e `customSparkConfig.properties` para defaults adicionais de submissão.

Jupyter

Quando `tdp-spark.integration.jupyter.enabled: true`, o chart renderiza um ConfigMap de integração para ambientes Jupyter.
Esse ConfigMap contém um `spark-defaults.conf` com os pares definidos em `tdp-spark.integration.jupyter.sparkConfig` e um script de preparação de ambiente para notebooks.

Exemplo de habilitação:

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-spark -n <NAMESPACE> --set tdp-
spark.integration.jupyter.enabled=true
```

© Copyright Tecnisys 2026. All rights reserved.

Modelo de valores:

```
tdp-spark:
  integration:
    jupyter:
      enabled: true
      sparkConfig:
        "spark.master": "spark://<SPARK_MASTER_SERVICE>.
<NAMESPACE>.svc.cluster.local:7077"
        "spark.serializer": "org.apache.spark.serializer.KryoSerializer"
        "spark.driver.bindAddress": "0.0.0.0"
```

O valor de `spark.master` deve apontar para o Service do master Spark visto a partir do namespace onde o cliente Jupyter executa.
Se Jupyter e Spark estiverem em namespaces diferentes, valide DNS, NetworkPolicy e portas antes de submeter notebooks.

Airflow

O bloco `tdp-spark.integration.airflow.sparkConfig` guarda defaults de cliente Spark para ambientes Airflow.
Use esse bloco para manter valores como master, memória de driver, memória de executor e número de cores em um ponto versionado de configuração.

Modelo de valores:

```
tdp-spark:
  integration:
    airflow:
      enabled: true
      sparkConfig:
        "spark.master": "spark://<SPARK_MASTER_SERVICE>.
<NAMESPACE>.svc.cluster.local:7077"
        "spark.driver.memory": "1g"
        "spark.executor.memory": "2g"
        "spark.executor.cores": "1"
```

No Airflow, alinhe esses valores ao operador usado para submeter jobs Spark.
O chart Spark não substitui a configuração do DAG; ele fornece defaults e convenções

© Copyright Tecnisys 2026. All rights reserved.

para o cliente.

Object Storage S3/S3A

S3/S3A é o caminho usual para integrar Spark com armazenamento compatível, como Ozone S3 Gateway, MinIO ou outro endpoint S3.
As chaves podem aparecer em `spark.sparkConf`, `tdp-spark.hadoopConfig` ou `tdp-spark.customSparkConfig.properties`.

Modelo de propriedades em `spark.sparkConf`:

```
spark:
  sparkConf:
    "spark.hadoop.fs.s3a.endpoint": "http://<S3_ENDPOINT>:<S3_PORT>"
    "spark.hadoop.fs.s3a.access.key": "<ACCESS_KEY>"
    "spark.hadoop.fs.s3a.secret.key": "<SECRET_KEY>"
    "spark.hadoop.fs.s3a.path.style.access": "true"
```

Credenciais e endpoints sensíveis devem ser tratados como valores de ambiente, arquivo privado de valores ou Secret gerenciado pela plataforma.
Para boas práticas, consulte Segurança - Spark.

Delta Lake

O bloco `tdp-spark.deltaLake` é opcional e organiza valores relacionados a Delta Lake. Habilitar esse bloco não altera automaticamente `spark.sparkConf`; forneça propriedades Spark, dependências e configurações de armazenamento conforme a imagem Spark usada no ambiente.

Modelo mínimo de ativação do bloco:

```
tdp-spark:
  deltaLake:
    enabled: true
```

Use `tdp-spark.customSparkConfig.properties` para declarar defaults adicionais quando a imagem exigir pacotes, extensões ou catálogos.

© Copyright Tecnisys 2026. All rights reserved.

Iceberg

O bloco `tdp-spark.iceberg` é opcional e organiza valores relacionados a Apache Iceberg. Quando usar Hive Metastore e S3/S3A, configure também o catálogo e o warehouse nas propriedades Spark.

Modelo de configuração:

```
spark:
  sparkConf:
    "spark.sql.catalog.iceberg": "org.apache.iceberg.spark.SparkCatalog"
    "spark.sql.catalog.iceberg.type": "hive"
    "spark.sql.catalog.iceberg.uri": "thrift://<HIVE_METASTORE_SERVICE>.
<NAMESPACE>.svc.cluster.local:9083"
    "spark.sql.catalog.iceberg.warehouse": "s3a://<S3_BUCKET>/hive"
    "spark.sql.extensions":
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
```

Esse exemplo é um modelo estrutural.
Valide bibliotecas, versão do Spark, catálogo e metastore conforme a imagem e os serviços instalados no ambiente.

Validação

Confirme os Services do release Spark:

```
kubectl -n <NAMESPACE> get svc -l app.kubernetes.io/instance=<RELEASE_NAME>
```

Quando a integração Jupyter estiver habilitada, verifique o ConfigMap gerado:

```
kubectl -n <NAMESPACE> get configmap tdp-spark-jupyter-integration
```

Para validar o acesso ao master a partir de outro namespace, execute um teste de resolução e porta com uma imagem de depuração aprovada no seu ambiente.
O endpoint esperado segue o formato `spark://<SPARK_MASTER_SERVICE>.<NAMESPACE>.svc.cluster.local:7077`.

© Copyright Tecnisys 2026. All rights reserved.

Superset

CHART

Version 3.0.1 | Type application | AppVersion 5.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

O que é o Apache Superset?

O Apache Superset é a camada de visualização de dados do TDP.
É onde analistas e gestores criam dashboards, exploram tabelas e geram gráficos sem precisar escrever SQL diretamente — embora o SQL também seja suportado para usuários avançados.

No TDP Kubernetes, o Superset se conecta ao **Trino** e ao **ClickHouse** como fontes de dados.
O Trino fornece acesso federado a todos os dados do data lake; o ClickHouse fornece queries analíticas de alta velocidade.
O Superset é o frontend que torna esses dados acessíveis a todos na organização.

> **PARA SABER MAIS**
>
> Consulte Apache Superset — Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

Os valores do Apache Superset devem ficar sob a chave `tdp-superset:` (alias do subchart upstream no `Chart.yaml`). Configurações específicas da plataforma TDP ficam sob `TDP-Settings:`.

```
TDP-Settings:
  externalDatabase:
    enabled: false
```

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-superset:
  supersetNode:
    replicaCount: 1
```

Arquitetura

O chart `tdp-superset` implanta os seguintes componentes:

Componente	Função
Superset Web	Interface web e API (Flask/Python)
Celery Workers	Processam queries assíncronas e caches em background
Celery Beat	Agendador de tarefas periódicas (refresh de dashboards, alertas)
Redis (embutido)	Broker de mensagens para o Celery e cache de resultados
PostgreSQL (interno ou externo)	Banco de metadados do Superset (dashboards, usuários, datasources)
Jobs auxiliares TDP	Bootstrap de banco externo e limpeza de Secret quando habilitados

O Superset **não** armazena dados de análise — ele armazena apenas as definições de dashboards, gráficos, conexões e usuários.
Para metadados, o chart pode usar o PostgreSQL embutido para instalações simples ou um PostgreSQL externo quando o ambiente exigir banco compartilhado.

Visão geral

Propriedade	Valor
Chart	`tdp-superset`

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
Versão do Superset	5.0.0
Versão do Chart	3.0.1
Registry	`oci://registry.tecnisys.com.br/tdp/charts/tdp-superset`

Páginas relacionadas

- Segurança — Superset: `SUPERSET_SECRET_KEY`, credenciais, LDAP e boas práticas.
- Integrações — Superset: importação de datasources Trino/ClickHouse.
- Exposição externa — Superset: configuração de Ingress e cuidado com recursos duplicados.

Pré-requisitos

Antes de instalar ou customizar o chart `tdp-superset`, confirme:

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível para volumes persistentes, principalmente quando usar PostgreSQL embutido.
- Registry OCI da Tecnisys acessível pelo ambiente de instalação.
- Uma `SUPERSET_SECRET_KEY` forte definida de forma segura.
- Se usar PostgreSQL externo: instância acessível a partir do cluster e Secret administrativo no padrão esperado pelo chart.
- Se usar auto-import de datasources: Trino/ClickHouse e as respectivas credenciais acessíveis durante a inicialização.

Instalação

Instalação

Exemplo

© Copyright Tecnisys 2026. All rights reserved.

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-superset -n <NAMESPACE> --create-namespace --timeout 10m
```

Exemplo:

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-superset --namespace <NAMESPACE> --values values.yaml
```

Antes da primeira instalação, defina onde o Superset vai guardar seus metadados. Use o PostgreSQL embutido para uma instalação inicial simples, ou configure PostgreSQL externo com `TDP-Settings.externalDatabase` quando o ambiente já tiver banco compartilhado.
Defina a `SUPERSET_SECRET_KEY` antes da instalação — ela não pode ser alterada sem planejamento após o primeiro deploy.

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão no chart
`tdp-superset.enabled`	Habilita o Superset	`true`
`tdp-superset.fullnameOverride`	Nome base dos recursos	`tdp-superset`
`tdp-superset.image.repository`	Imagem Superset TDP com drivers e dependências	Imagem TDP Superset
`tdp-superset.extraSecretEnv.SUPERSET_SECRET_KEY`	Chave secreta do Superset (obrigatória)	placeholder

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão no chart
`global.compatibility.openshift.adaptSecurityContext`	Adaptação automática para OpenShift	`force`
`supersetPostgresqlPassword`	Senha do usuário DB do Superset (Secret gerido pelo chart)	`""`
`TDP-Settings.externalDatabase.enabled`	Ativa job de bootstrap `db-create-job`	`false`
`TDP-Settings.externalDatabase.externalSecret.releaseName`	Nome do Secret PostgreSQL admin	`tdp-postgresql`
`TDP-Settings.gateway.ingress.enabled`	Renderiza Ingress TDP	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Renderiza HTTPRoute Gateway API	`false`
`tdp-superset.postgresqlSecrets.name`	Nome do Secret da senha do DB Superset	`""`
`tdp-superset.postgresqlSecrets.passwordKey`	Chave da senha dentro do Secret	`superset-postgresql-password`
`tdp-superset.supersetNode.connections.db_host`	Host do PostgreSQL de metadados	`<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local`
`tdp-superset.service.port`	Porta da interface web/API	`8088`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão no chart
`tdp-superset.datasources.autoImport.enabled`	Auto-import de datasources no init	`false`
`tdp-superset.datasources.clickhouse.enabled`	Bloco ClickHouse	`false`
`tdp-superset.datasources.trino.enabled`	Bloco Trino	`false`

> **OPENSHIFT**
>
> Em ambientes OpenShift, o chart aplica automaticamente `global.compatibility.openshift.adaptSecurityContext: force` e usa `tdp-superset.postgresql.postgresqlDataDir: /bitnami/postgresql/data/pgdata` para evitar erros de permissão na raiz do volume do PostgreSQL embutido. Nenhuma configuração adicional é necessária para compatibilidade básica.

Detalhes de configuração

PostgreSQL de metadados

Com `TDP-Settings.externalDatabase.enabled=false` (padrão), o chart usa o subchart PostgreSQL embutido.
Com `enabled=true`, o Superset conecta a um PostgreSQL externo e `tdp-superset.postgresql.enabled` deve ser `false`.

Nesse caso, o chart cria banco e usuário via Helm hook — o job `db-create-job` — quando simultaneamente:

- `TDP-Settings.externalDatabase.enabled=true`
- `tdp-superset.postgresql.enabled=false`
- `tdp-superset.supersetNode.connections.db_type=postgresql`

O job lê a senha do administrador PostgreSQL de um Secret nomeado `<RELEASE_NAME>`, chave `postgres-password`, no padrão TDP.

© Copyright Tecnisys 2026. All rights reserved.

Esse Secret deve existir no namespace antes do upgrade.

```
tdp-superset:
  postgresql:
    enabled: false
  supersetNode:
    connections:
      db_type: postgresql
      db_host: <POSTGRESQL_SERVICE>.<NAMESPACE>.svc.cluster.local
      db_name: <DATABASE_NAME>
      db_user: <DATABASE_USER>
```

Datasources: ClickHouse e Trino

O Superset precisa conhecer as fontes de dados antes de criar gráficos ou dashboards. Essa configuração — chamada *datasource* — registra a conexão com nome, host, porta e credenciais no banco de metadados do Superset.

O chart pode fazer esse registro automaticamente durante a primeira inicialização, por meio do job de init.
Para isso, dois controles precisam estar habilitados:

- `tdp-superset.datasources.autoImport.enabled: true` — liga o mecanismo de auto-import no job de init.
- `tdp-superset.datasources.clickhouse.enabled: true` e/ou `tdp-superset.datasources.trino.enabled: true` — habilita o bloco de cada datasource individualmente.

Se `autoImport.enabled` estiver `false` (padrão), o Superset sobe sem datasources pré-cadastrados.
Nesse caso, o administrador registra as conexões manualmente pela interface após o primeiro login.

Para exemplos detalhados de configuração de cada datasource, parâmetros disponíveis e dependências de ordem, consulte Integrações — Superset.

Réplicas e recursos

O Superset é composto por processos com funções distintas e configurações de escala independentes.

© Copyright Tecnisys 2026. All rights reserved.



Entender o papel de cada um ajuda a dimensionar corretamente o ambiente.

O **nó web** (`supersetNode`) responde às requisições HTTP e à API do Superset.
Múltiplas réplicas aumentam a disponibilidade e distribuem carga de acesso simultâneo.

Os **workers Celery** (`supersetWorker`) executam tarefas assíncronas: consultas SQL de longa duração, atualização de cache e disparo de alertas.
Aumente a quantidade de workers quando o volume de consultas assíncronas for alto ou houver fila de espera.

O **Celery Beat** (`supersetCeleryBeat`) é o agendador de tarefas periódicas — refresh automático de dashboards e alertas programados.
Uma única instância é suficiente; não é projetado para escala horizontal.

O **Redis** embutido é o broker de mensagens que conecta o nó web aos workers Celery e armazena o cache de resultados.
Para ambientes maiores, considere um Redis externo dedicado.

`requests` definem o recurso mínimo garantido ao pod; `limits` definem o máximo permitido.
Dimensione workers com `requests` adequados ao tamanho das consultas que serão processadas.

Parâmetro	Descrição
`tdp-superset.supersetNode.replicas.enabled`	Habilita múltiplas réplicas do nó web
`tdp-superset.supersetNode.replicas.replicaCount`	Quantidade de réplicas do nó web
`tdp-superset.supersetWorker.replicas.enabled`	Habilita múltiplas réplicas de workers Celery
`tdp-superset.supersetWorker.replicas.replicaCount`	Quantidade de workers Celery

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição
`tdp-superset.supersetCeleryBeat.enabled`	Habilita o agendador Celery Beat
`tdp-superset.resources`	Recursos do nó web (requests/limits)
`tdp-superset.supersetWorker.resources`	Recursos dos workers Celery (requests/limits)
`tdp-superset.init.resources`	Recursos do job de inicialização
`tdp-superset.redis.enabled`	Redis embutido
`tdp-superset.postgresql.postgresqlDataDir`	Diretório de dados do PostgreSQL embutido

Acesso & Segurança

Segurança

A segurança do Superset abrange a chave de sessão, o usuário admin inicial, a autenticação LDAP e o Secret do PostgreSQL externo. Consulte Segurança — Superset para esses parâmetros, mapeamento de grupos, boas práticas e solução de problemas.

Exposição externa

Ingress e Gateway API são alternativas mutuamente exclusivas controladas por `TDP-Settings.gateway`.
Deixe `tdp-superset.ingress.enabled=false` quando usar o Ingress TDP, para evitar Ingress duplicado pelo subchart upstream.

Consulte Exposição externa — Superset para a configuração de Ingress e o cuidado com recursos duplicados.

Acesso local

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> port-forward svc/<SERVICE_NAME> 8088:8088
```

Exemplo:

```
kubectl -n tdp-project port-forward svc/tdp-superset 8088:8088
```

Abra `http://127.0.0.1:8088`.
Credenciais iniciais vêm do bloco `tdp-superset.init` no seu arquivo de valores.
Altere-as após o primeiro login.

Solução de problemas

Solução de problemas

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

Desinstalação

Desinstalação

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Para a lista completa de parâmetros, execute `helm show values` para a versão do chart que instalou.

© Copyright Tecnisys 2026. All rights reserved.

Superset – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 5.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral

O chart `tdp-superset` concentra a segurança em três frentes principais.
A primeira é a `SUPERSET_SECRET_KEY`, usada pelo Superset para proteger sessões, cookies e dados sensíveis persistidos no banco de metadados.
A segunda é a proteção das credenciais do PostgreSQL de metadados, incluindo o Secret administrado pelo chart e o Secret administrativo usado no bootstrap externo.
A terceira é a autenticação LDAP, configurada pelo bloco `tdp-superset.ldap` e integrada ao mapeamento de roles do Superset.

A ideia prática é simples: a chave protege o estado interno da aplicação, o PostgreSQL guarda os metadados e o LDAP controla quem entra.
Quando LDAP ou PostgreSQL externo estiverem ativos, trate Secrets, senhas e dependências de rede como pré-requisitos do deploy.

Modos de operação

O chart pode operar com usuários internos do Superset ou com autenticação LDAP. Essa diferença é controlada principalmente por `tdp-superset.ldap.enabled` e pelos valores de inicialização em `tdp-superset.init`.

Modo	`tdp-superset.ldap.enabled`	Autenticação	Usuários e roles
Usuários internos	`false`	Login gerenciado pelo Superset	Usuário inicial criado pelo bloco `tdp-superset.init`

© Copyright Tecnisys 2026. All rights reserved.

Modo	`tdp-superset.ldap.enabled`	Autenticação	Usuários e roles
LDAP	`true`	Login via diretório LDAP	Usuários registrados conforme `ldap.registration.*` e roles mapeadas por `ldap.roleMapping`

Com LDAP desabilitado, o usuário administrador inicial vem do bloco `tdp-superset.init`.
Esse modo é útil para instalação simples, testes controlados ou ambientes onde o acesso é restrito por outra camada.

Com LDAP habilitado, o Superset usa o diretório corporativo como provedor de autenticação.
Nesse modo, o Secret com a senha do usuário de bind precisa existir antes do deploy, e as roles devem ser planejadas para evitar concessão ampla de permissões.

Pré-requisitos

- `SUPERSET_SECRET_KEY` forte definida antes do primeiro deploy.
- Secret de bind LDAP criado no namespace quando `tdp-superset.ldap.enabled: true`.
- Servidor LDAP acessível a partir do namespace do Superset quando LDAP for usado.
- PostgreSQL de metadados acessível e Secret administrativo disponível quando usar banco externo.
- Políticas de rede e exposição configuradas para limitar acesso à interface do Superset.

SUPERSET_SECRET_KEY

A chave `tdp-superset.extraSecretEnv.SUPERSET_SECRET_KEY` é **obrigatória**.
O Superset usa essa chave para criptografar sessões, cookies e dados sensíveis persistidos no banco de metadados.
Gere um valor com entropia adequada e injete via Secret do Kubernetes ou ferramenta

© Copyright Tecnisys 2026. All rights reserved.

de gestão de segredos.
Não versione essa chave em texto claro.

Modelo para gerar um valor:

```
openssl rand -base64 42
```

Rotacionar essa chave sem planejamento invalida sessões ativas e dados já cifrados com o valor anterior.
Trate-a com o mesmo rigor de uma senha de infraestrutura.

PostgreSQL de metadados

O chart gerencia um Secret para a senha do usuário de banco de metadados do Superset.
O nome e a chave do Secret são controlados por `tdp-superset.postgresqlSecrets.name` e `tdp-superset.postgresqlSecrets.passwordKey`.

Quando `TDP-Settings.externalDatabase.enabled=true`, o job de bootstrap gera a senha automaticamente e a armazena no Secret.
Modelo para consultar o Secret gerado:

```
kubectl -n <NAMESPACE> get secret <RELEASE_NAME>-postgresql-secrets -o
jsonpath='{.data.superset-postgresql-password}' | base64 -d
```

Para PostgreSQL externo, o job `db-create-job` lê a senha do administrador em um Secret com o nome `<RELEASE_NAME>` e a chave `postgres-password`.
Esse Secret precisa existir no namespace antes do hook de instalação ou atualização.
O nome do Secret é definido por `TDP-Settings.externalDatabase.externalSecret.releaseName` (padrão `tdp-postgresql`).

LDAP habilitado

LDAP é usado pelo Superset como provedor de autenticação.
No chart, habilitar LDAP substitui o uso exclusivo do banco interno de usuários por login no diretório corporativo.

© Copyright Tecnisys 2026. All rights reserved.

Antes de habilitar LDAP, crie o Secret com a senha do usuário de bind no namespace do Superset:

```
kubectl -n <NAMESPACE> create secret generic tdp-superset-ldap-secret --from-literal=bind-password=<LDAP_BIND_PASSWORD>
```

Modelo de configuração:

```
tdp-superset:
  ldap:
    enabled: true
    server:
      url: "ldap://<LDAP_HOST>:389"
      useTls: "False"
    bind:
      user: "cn=<LDAP_BIND_USER>,cn=users,cn=accounts,dc=<LDAP_DOMAIN>,dc=com"
      passwordSecret:
        name: "tdp-superset-ldap-secret"
        key: "bind-password"
    search:
      base: "cn=users,cn=accounts,dc=<LDAP_DOMAIN>,dc=com"
      filter: "(objectClass=person)"
      uidField: "uid"
    userAttributes:
      firstname: "givenName"
      lastname: "sn"
      email: "mail"
    registration:
      enabled: "True"
      defaultRole: "Public"
    roleMapping:
      "cn=superset_admins,cn=groups,cn=accounts,dc=<LDAP_DOMAIN>,dc=com":
["Admin"]
      "cn=superset_users,cn=groups,cn=accounts,dc=<LDAP_DOMAIN>,dc=com":
["Gamma", "Alpha"]
    groupField: "memberOf"
    roleSyncAtLogin: "True"
```

Substitua os placeholders antes de instalar.
`<LDAP_HOST>` é o endereço do LDAP visto de dentro do cluster.

© Copyright Tecnisys 2026. All rights reserved.



`<LDAP_BIND_USER>` é a conta usada pelo Superset para pesquisar usuários e grupos.
`<LDAP_DOMAIN>` representa os componentes do DN do diretório.
`<LDAP_BIND_PASSWORD>` deve ser armazenado apenas no Secret.

Para desabilitar LDAP e voltar ao modo de usuários internos, defina `tdp-superset.ldap.enabled: false` e remova a configuração LDAP específica.
Depois do upgrade, valide se o login interno esperado ainda existe e se as políticas de exposição continuam restritas.

Mapeamento de grupos e roles

O parâmetro `ldap.roleMapping` mapeia grupos do diretório para roles do Superset.
A chave é o DN completo do grupo no LDAP; o valor é uma lista de roles do Superset.

Roles nativas do Superset: `Admin`, `Alpha`, `Gamma`, `Public`.
Use `Gamma` como `defaultRole` para acesso somente leitura a dashboards.
Use `Alpha` para usuários que criam gráficos e datasets.
Use `Admin` apenas para grupos administrativos controlados.

Com `ldap.roleSyncAtLogin: "True"`, as roles do usuário são recalculadas a cada login com base nos grupos do LDAP.
Mudanças no diretório, como adicionar ou remover grupos, se refletem no Superset sem intervenção manual.

Parâmetros principais

Parâmetro	Papel	Padrão	Quando alterar
`tdp-superset.extraSecretEnv.SUPERSET_SECRET_KEY`	Chave criptográfica da aplicação	placeholder	Sempre definir antes do primeiro deploy
`tdp-superset.init`	Criação do usuário admin inicial	valores do chart	Instalações sem LDAP ou bootstrap inicial
`tdp-superset.postgresqlSecrets.nam`	Nome do Secret da senha do	`""`	Quando usar Secret próprio

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Papel	Padrão	Quando alterar
`e`	banco de metadados		
`tdp-superset.postgresqlSecrets.passwordKey`	Chave da senha dentro do Secret	`superset-postgresql-password`	Quando o Secret usar outra chave
`TDP-Settings.externalDatabase.enabled`	Habilita bootstrap de PostgreSQL externo	`false`	Ambientes com banco compartilhado
`TDP-Settings.externalDatabase.externalSecret.releaseName`	Nome do Secret admin do PostgreSQL	`tdp-postgresql`	Quando o Secret admin tiver outro nome
`tdp-superset.ldap.enabled`	Habilita autenticação LDAP	`false`	Ambientes com autenticação corporativa
`tdp-superset.ldap.server.url`	URL do servidor LDAP	`ldap://<LDAP_HOST>:389`	Quando usar outro host, porta ou LDAPS
`tdp-superset.ldap.bind.passwordSecret.*`	Secret com senha do usuário de bind	`tdp-superset-ldap-secret` / `bind-password`	Quando usar outro Secret ou chave
`tdp-superset.ldap.search.*`	Base, filtro e atributo de login	valores LDAP do ambiente	Sempre ajustar ao diretório corporativo
`tdp-superset.ldap.registration.*`	Auto-registro e role padrão	`True` / `Public`	Ao controlar perfil inicial de usuários

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Papel	Padrão	Quando alterar
`tdp-superset.ldap.roleMapping`	Mapeia grupos LDAP para roles	`{}`	Ambientes com RBAC por grupos
`tdp-superset.ldap.roleSyncAtLogin`	Sincroniza roles no login	`True`	Quando roles devem refletir alterações no LDAP

Solução de problemas

Situação	O que verificar
Pods não sobem	`SUPERSET_SECRET_KEY` definida e sem espaços extras
Init job falha	Host DB, senha e Secret admin `<RELEASE_NAME>` com a chave `postgres-password`
Login com usuários internos falha	Bloco `tdp-superset.init` com usuário admin no arquivo de valores
Login LDAP falha	URL LDAP, DN de bind, Secret `tdp-superset-ldap-secret` e conectividade
Usuário LDAP não encontrado	`ldap.search.base`, `ldap.search.filter` e `ldap.search.uidField`
Role errada após login	`ldap.roleMapping`, DN dos grupos e `ldap.roleSyncAtLogin: "True"`
LDAP habilitado mas sem efeito	Confirmar `ldap.enabled: true` e Secret de bind criado antes do deploy

Ao trocar de usuários internos para LDAP, crie o Secret de bind antes do `helm upgrade`. Depois valide login, atributos de usuário e mapeamento de roles.

© Copyright Tecnisys 2026. All rights reserved.



Ao trocar de LDAP para usuários internos, confirme que existe usuário admin local válido e revise a exposição da interface.

Boas práticas

- Não versionar a senha de bind LDAP nem a `SUPERSET_SECRET_KEY`.
- Rotacionar credenciais após o primeiro deploy em produção.
- Use `Public` ou `Gamma` como `defaultRole` para evitar acesso administrativo inadvertido a novos usuários LDAP.
- Prefira `secretKeyRef` em `extraEnvRaw` em vez de valores literais em `extraEnv`.
- Restrinja a exposição do serviço por `NetworkPolicy` e use Ingress apenas quando necessário.
- Para LDAPS (porta 636), ajuste `ldap.server.url` para `ldaps://` e `ldap.server.useTls: "True"`.

© Copyright Tecnisys 2026. All rights reserved.

Superset – Integração

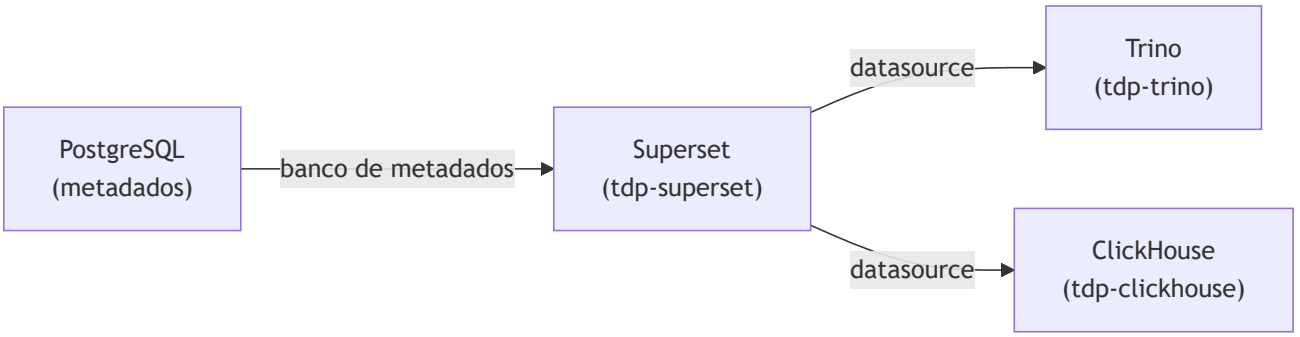
CHART

Version 3.0.1 | Type application | AppVersion 5.0.0

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral



O chart `tdp-superset` pode usar um PostgreSQL externo (`tdp-postgresql`) como banco de metadados — consulte Configuração do Superset.

O chart `tdp-superset` registra datasources para Trino e ClickHouse automaticamente durante a inicialização quando `tdp-superset.datasources.autoImport.enabled: true` e os blocos de cada datasource estão habilitados.

O job de importação é executado em toda instalação e atualização (`post-install,post-upgrade`). Ele realiza um **upsert por UUID** — conexões existentes são atualizadas, não duplicadas. Quando `autoImport.enabled: false`, o ConfigMap de datasources não é criado e a montagem de volume é ignorada silenciosamente.
Para que a importação funcione, os serviços de dados precisam estar acessíveis no cluster antes do término do job.
Se o auto-import falhar ou os serviços não estiverem disponíveis, os datasources podem ser cadastrados manualmente pela interface do Superset após o deploy.

ClickHouse

© Copyright Tecnisys 2026. All rights reserved.

O bloco `tdp-superset.datasources.clickhouse` configura a conexão com um cluster ClickHouse.

Exemplo

```
tdp-superset:
  datasources:
    autoImport:
      enabled: true
    clickhouse:
      enabled: true
      name: "ClickHouse TDP"
      host: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: "8123"
      database: "default"
      username: "<USERNAME>"
      password: "<PASSWORD>"
      uuid: "92122244-b262-44a6-a2ed-e49a05860dda"
```

Campo	Descrição
`host`	Endereço interno do serviço ClickHouse no formato `<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local`
`port`	Porta HTTP do ClickHouse. O padrão é `8123`
`database`	Banco de dados padrão para o data-source. Use `default` se não houver banco específico
`username`	Usuário com acesso de leitura ao ClickHouse
`password`	Senha do usuário. Não versione em texto claro

© Copyright Tecnisys 2026. All rights reserved.

Campo	Descrição
`uuid`	Identificador estável do datasource. Não altere após a primeira criação no ambiente

Trino

O bloco `tdp-superset.datasources.trino` configura a conexão com um cluster Trino.

Exemplo

```
tdp-superset:
  datasources:
    autoImport:
      enabled: true
    trino:
      enabled: true
      name: "Trino TDP"
      host: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: "8080"
      catalog: ""
      username: "<USERNAME>"
      password: ""
      uuid: "8a92c01b-2d6e-4fc1-a302-b6a5f0e56789"
```

Campo	Descrição
`host`	Endereço interno do serviço Trino no formato `<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local`
`port`	Porta do Trino. Use `8080` para HTTP sem LDAP
`catalog`	Catálogo Trino padrão para o data-source. Deixe vazio para que o Superset

© Copyright Tecnisys 2026. All rights reserved.

Campo	Descrição
	liste todos os catálogos disponíveis
`username`	Usuário Trino. Para ambientes sem LDAP, qualquer valor é aceito
`password`	Senha Trino. Deixe vazio quando LDAP não estiver ativo
`uuid`	Identificador estável do datasource. Não altere após a primeira criação no ambiente

Combinar ClickHouse e Trino

Para importar ambos os datasources no mesmo deploy:

Exemplo

```
tdp-superset:
  datasources:
    autoImport:
      enabled: true
    clickhouse:
      enabled: true
      name: "ClickHouse TDP"
      host: "<CLICKHOUSE_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: "8123"
      database: "default"
      username: "<USERNAME>"
      password: "<PASSWORD>"
      uuid: "92122244-b262-44a6-a2ed-e49a05860dda"
    trino:
      enabled: true
      name: "Trino TDP"
      host: "<TRINO_SERVICE>.<NAMESPACE>.svc.cluster.local"
      port: "8080"
      catalog: ""
      username: "<USERNAME>"
```

© Copyright Tecnisys 2026. All rights reserved.

```
      password: ""
      uuid: "8a92c01b-2d6e-4fc1-a302-b6a5f0e56789"
```

UUID do datasource

O `uuid` identifica o datasource de forma estável entre deploys.
Não altere o UUID após a primeira criação em um ambiente existente, pois dashboards e gráficos que referenciam esse datasource usam o UUID como chave.
Para um ambiente novo, o UUID padrão do chart pode ser mantido ou substituído por um novo gerado localmente.

Para gerar um UUID novo localmente, use uma ferramenta de UUID v4.
Em ambientes com `uuidgen`:

```
uuidgen
```

Outra opção, usando Python:

```
python -c "import uuid; print(uuid.uuid4())"
```

Use o valor gerado no campo `uuid` do datasource antes do primeiro deploy do ambiente.

Dependência de ordem

O auto-import ocorre durante o job de inicialização do Superset.
Certifique-se de que os serviços Trino e ClickHouse estejam em execução e acessíveis antes de iniciar o deploy do Superset com `autoImport.enabled: true`.
Se o job de init for executado antes dos datasources estarem prontos, o import falhará e precisará ser refeito manualmente.

Validação de acessibilidade

Antes de habilitar o auto-import, confirme se os Services de Trino e ClickHouse existem no namespace esperado:

```
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> get svc
```

Valide também a conectividade a partir de dentro do cluster.
Para Trino, o endpoint `/v1/info` confirma que o serviço HTTP está respondendo:

```
kubectl -n <NAMESPACE> run trino-check --rm -it --image=curlimages/curl --
restart=Never -- curl -sS http://<TRINO_SERVICE>.
<NAMESPACE>.svc.cluster.local:8080/v1/info
```

Para ClickHouse, o endpoint `/ping` deve retornar `Ok`:

```
kubectl -n <NAMESPACE> run clickhouse-check --rm -it --image=curlimages/curl --
restart=Never -- curl -sS http://<CLICKHOUSE_SERVICE>.
<NAMESPACE>.svc.cluster.local:8123/ping
```

Se a importação falhar, consulte os jobs e logs de inicialização do Superset:

```
kubectl -n <NAMESPACE> get jobs
kubectl -n <NAMESPACE> logs job/<JOB_NAME>
```

Cadastro manual pela interface

Quando o auto-import estiver desabilitado, falhar no init ou a fonte de dados for criada depois do deploy, cadastre o datasource pela interface do Superset.
Na UI, acesse `Settings > Database Connections` e crie uma nova conexão.

Use os mesmos dados previstos no bloco do chart: nome do datasource, host, porta, usuário, senha e banco ou catálogo padrão.
Para ClickHouse, normalmente informe a porta HTTP `8123` e o banco de dados padrão.
Para Trino, informe a porta `8080` para HTTP sem LDAP, o usuário Trino e, quando necessário, o catálogo inicial.

Depois de salvar, use a ação de teste de conexão da própria interface antes de criar datasets, gráficos ou dashboards.

Segurança e credenciais

© Copyright Tecnisys 2026. All rights reserved.

Não versione senhas de datasources em texto claro.
Para ambientes de produção, injete as senhas via Secret do Kubernetes usando `secretKeyRef` em `extraEnvRaw`, ou use ferramenta de gestão de segredos.
Consulte Segurança — Superset para boas práticas de credenciais e configuração de LDAP.

© Copyright Tecnisys 2026. All rights reserved.

Trino

CHART

Version 3.0.1 | Type application | AppVersion 478

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

SUPORTE A RECURSOS

LDAP suportado | S3 suportado | Ingress suportado | Gateway API suportado

O que é o Trino?

O Trino é o motor de consulta SQL distribuida do TDP Kubernetes.
Ele consulta múltiplas fontes de dados com um único SQL, como ClickHouse, PostgreSQL, Hive, Iceberg, Kafka e dados em storage S3/Ozone, sem mover os dados para dentro do Trino.

O Trino não armazena dados próprios: ele é um **query engine federado**.
As fontes de dados são expostas por **catálogos** configurados no chart.

> **PARA SABER MAIS**
>
> Consulte Trino - Conceitos para uma visão completa da ferramenta, sua arquitetura e funcionamento.

Estrutura de valores (Helm)

Todos os valores do Trino devem ficar sob a chave `tdp-trino:` (alias da dependência no `Chart.yaml`).

Respeitar esse prefixo garante que o Helm aplique os valores no subchart correto:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-trino:
  server:
    workers: 3
  service:
    port: 8080
```

Arquitetura coordinator-worker

O chart `tdp-trino` implanta um coordinator e um conjunto de workers:

Componente	Função
Coordinator	Recebe a consulta SQL, cria o plano de execução e distribui o trabalho
Workers	Executam as etapas do plano em paralelo, processando dados nas fontes externas

O coordinator é único.
Para aumentar paralelismo, ajuste `tdp-trino.server.workers` ou os mecanismos de autoscaling dos workers.

Visão geral

Propriedade	Valor
Chart	`tdp-trino`
Versão do Trino	478
Versão do Chart	3.0.1
Registry	`oci://registry.tecnisys.com.br/tdp/charts/tdp-trino`

© Copyright Tecnisys 2026. All rights reserved.

Propriedade	Valor
AppVersion do chart	`478`

Páginas relacionadas

- Integrações - Trino: catálogos, conectores e dependências de fontes externas.
- Segurança - Trino: LDAP, HTTPS, certificados, Secrets e Ranger.
- Exposição externa - Trino: exposição HTTP/HTTPS via Ingress ou Gateway API.

Pré-requisitos

Antes de instalar ou customizar o chart `tdp-trino`, confirme:

- Kubernetes 1.32+, Red Hat OpenShift 4.19+ ou Rancher Manager 2.10.x+.
- Helm 3.2.0+.
- StorageClass disponível quando usar certificados, PVCs ou recursos que precisem de volume persistente.
- Registry OCI da Tecnisys acessível pelo ambiente de instalação.
- **Requisito de CPU:** Os nós workers do cluster devem suportar o conjunto de instruções **x86-64-v3** (Intel Haswell 2013+ ou AMD Zen 2+). O Trino 478+ exige as extensões AVX2, BMI1, BMI2 e FMA.
- Fontes externas acessíveis a partir do namespace do Trino quando usar catálogos adicionais.
- Metrics Server e `resources.requests` dos workers configurados para HPA; para KEDA, KEDA instalado e Prometheus disponível com métricas JMX do Trino.
- Certificados e Secrets preparados quando usar HTTPS, LDAP ou Ranger.

Instalação

Instalação

Exemplo

```
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-trino
-n <NAMESPACE> --create-namespace
```

Exemplo:

© Copyright Tecnisys 2026. All rights reserved.



```
helm upgrade tdp-trino oci://registry.tecnisys.com.br/tdp/charts/tdp-trino \
  --namespace tdp-project \
  --values values.yaml
```

Os nomes dos recursos derivam do release Helm por padrão.
Use `tdp-trino.fullnameOverride`, `tdp-trino.coordinatorNameOverride` e `tdp-trino.workerNameOverride` apenas quando precisar evitar conflitos de nomes, por exemplo ao implantar múltiplos releases no mesmo namespace.

Parâmetros principais

Parâmetros principais

Parâmetro	Descrição	Padrão no chart
`tdp-trino.enabled`	Habilita a implantação do Trino	`true`
`tdp-trino.image.tag`	Tag da imagem Trino	`478`
`tdp-trino.server.workers`	Quantidade inicial de workers	`2`
`tdp-trino.server.node.environment`	Rótulo de ambiente do cluster Trino	`environment`
`tdp-trino.service.type`	Tipo do Service Kubernetes	`ClusterIP`
`tdp-trino.service.port`	Porta HTTP do Service	`8080`
`tdp-trino.server.config.https.enabled`	Habilita HTTPS no Trino	`false`
`tdp-trino.server.config.authenticationType`	Tipo de autenticação	`""`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão no chart
`tdp-trino.catalogs`	Definições de catálogos/connectors	`tpch`, `tpcds`, `memory`
`tdp-trino.accessControl`	Configuração de controle de acesso	`{}`
`tdp-trino.server.autoscaling.enabled`	HPA para workers	`true`
`tdp-trino.server.keda.enabled`	KEDA para workers	`false`
`tdp-trino.jmx.enabled`	Habilita JMX	`true`
`tdp-trino.serviceMonitor.enabled`	Cria ServiceMonitor	`false`
`trinoCerts.pvc.enabled`	PVC para certificados	`false`
`trinoCerts.secret.enabled`	Secret com certificados	`false`
`trinoCerts.copyJob.enabled`	Job de cópia de certificados para PVC	`false`
`TDP-Settings.gateway.ingress.enabled`	Renderiza Ingress TDP	`false`
`TDP-Settings.gateway.gatewayApi.enabled`	Renderiza HTTPRoute Gateway API	`false`

Detalhes de configuração

Catálogos

© Copyright Tecnisys 2026. All rights reserved.



Catálogos são a forma como o Trino registra fontes de dados externas.
Cada catálogo define um connector e as propriedades de acesso necessárias para o Trino localizar schemas, tabelas e objetos dessa fonte.

Na configuração do chart, os catálogos ficam em `tdp-trino.catalogs`.
Cada chave vira um catálogo no Trino e passa a ser usada nas consultas no formato `catalog.schema.objeto`.

O chart já traz catálogos de referência para `tpch`, `tpcds` e `memory`.
Em produção, adicione os catálogos necessários ao seu ambiente, como ClickHouse, PostgreSQL, Hive, Iceberg ou Kafka.

Consulte Integrações - Trino para exemplos e pré-requisitos por connector.

Recursos de CPU e memória

Workers executam o processamento das consultas e normalmente precisam de mais recursos que o coordinator.
Ajuste os valores conforme a complexidade das consultas e o volume de dados.

Parâmetro	Descrição	Padrão no chart
`tdp-trino.coordinator.resources.requests.cpu`	CPU reservada para o coordinator	`1`
`tdp-trino.coordinator.resources.requests.memory`	Memória reservada para o coordinator	`2Gi`
`tdp-trino.coordinator.resources.limits.cpu`	Limite de CPU do coordinator	`2`
`tdp-trino.coordinator.resources.limits.memory`	Limite de memória do coordinator	`4Gi`
`tdp-trino.worker.resources.requests.cpu`	CPU reservada para cada worker	`1`

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Descrição	Padrão no chart
`tdp-trino.worker.resources.requests.memory`	Memória reservada para cada worker	`2Gi`
`tdp-trino.worker.resources.limits.cpu`	Limite de CPU de cada worker	`2`
`tdp-trino.worker.resources.limits.memory`	Limite de memória de cada worker	`4Gi`

```
tdp-trino:
  coordinator:
    resources:
      requests:
        cpu: "2"
        memory: "4Gi"
      limits:
        cpu: "4"
        memory: "8Gi"
  worker:
    resources:
      requests:
        cpu: "2"
        memory: "4Gi"
      limits:
        cpu: "4"
        memory: "8Gi"
```

Autoscaling e observabilidade

O chart suporta HPA nativo por `tdp-trino.server.autoscaling` e KEDA por `tdp-trino.server.keda`.
Não combine HPA nativo e KEDA para os mesmos workers.

O HPA nativo (`tdp-trino.server.autoscaling.enabled`) e o JMX (`tdp-trino.jmx.enabled`) já vêm habilitados por padrão no chart, com `maxReplicas: 5`, `targetCPUUtilizationPercentage: 50` e `targetMemoryUtilizationPercentage: 80`.

© Copyright Tecnisys 2026. All rights reserved.



Para usar KEDA no lugar do HPA nativo, defina `tdp-trino.server.autoscaling.enabled: false` e `tdp-trino.server.keda.enabled: true`.

Para KEDA, o cluster deve ter KEDA instalado, Prometheus disponível com métricas JMX do Trino e `tdp-trino.serviceMonitor.enabled: true` quando usar Prometheus Operator.

O HPA usa métricas de CPU e memória do Kubernetes para ajustar a quantidade de workers.
O KEDA é usado quando a escala precisa responder a métricas externas; neste chart, o exemplo usa Prometheus consultando métricas JMX do Trino.
O JMX expõe métricas internas do Trino e o ServiceMonitor orienta o Prometheus Operator a coletá-las.

Exemplo de HPA:

```
tdp-trino:
  server:
    workers: 2
    autoscaling:
      enabled: true
      maxReplicas: 10
      targetCPUUtilizationPercentage: 70
      targetMemoryUtilizationPercentage: 80
      behavior:
        scaleDown:
          stabilizationWindowSeconds: 300
        scaleUp:
          stabilizationWindowSeconds: 0

  worker:
    resources:
      requests:
        cpu: "2"
        memory: "8Gi"
      limits:
        cpu: "4"
        memory: "16Gi"
```

Exemplo de KEDA:

© Copyright Tecnisys 2026. All rights reserved.

```
tdp-trino:
  server:
    autoscaling:
      enabled: false

    keda:
      enabled: true
      pollingInterval: 30
      cooldownPeriod: 300
      minReplicaCount: 1
      maxReplicaCount: 10
      triggers:
        - type: prometheus
          metricType: Value
          metadata:
            serverAddress: "http://prometheus-
server.monitoring.svc.cluster.local"
            threshold: "1"
            metricName: required_workers
            query: >-

sum(avg_over_time(trino_execution_ClusterSizeMonitor_RequiredWorkers{
                service="tdp-trino"
              }[1m]))

  jmx:
    enabled: true
    exporter:
      enabled: true

  serviceMonitor:
    enabled: true
```

Acesso & Segurança

Segurança

O chart ajusta HTTPS e autenticação conforme `ldap.enabled`:

© Copyright Tecnisys 2026. All rights reserved.

Modo	Comportamento
`ldap.enabled: false` (padrão)	Usa apenas HTTP na porta 8080, sem autenticação obrigatória, sem volumes de certificado e sem shared secrets.
`ldap.enabled: true`	Habilita HTTPS na porta 8443, autenticação `PASSWORD`, montagem de certificados, shared secrets e integração LDAP completa.

Ranger usa `tdp-trino.accessControl` e arquivos adicionais no coordinator, além de certificados quando o ambiente exige HTTPS.
Consulte Segurança - Trino para os exemplos completos.

Exposição externa

Ingress e Gateway API são mutuamente exclusivos, controlados por `TDP-Settings.gateway.*`. O Trino é exposto na porta `8080` (ou `8443` com HTTPS/LDAP).

Para os exemplos completos de values (Ingress e Gateway API), consulte Exposição externa.

Acesso local

```
kubectl -n <NAMESPACE> port-forward service/<RELEASE_NAME> 8080:8080
```

Com LDAP/HTTPS, use a porta 8443 conforme a configuração aplicada.

Solução de problemas

Solução de problemas

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get svc
kubectl -n <NAMESPACE> get events --sort-by=.lastTimestamp
```

© Copyright Tecnisys 2026. All rights reserved.

Desinstalação

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```

Para a lista completa de parâmetros, consulte o `values.yaml` do chart ou execute `helm show values` para a versão instalada.

© Copyright Tecnisys 2026. All rights reserved.

Trino – Segurança

CHART

Version 3.0.1 | Type application | AppVersion 478

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral

O chart `tdp-trino` concentra a segurança em três frentes principais.
A primeira é a autenticação LDAP, que muda o acesso do Trino de HTTP sem autenticação para HTTPS com `PASSWORD` authentication.
A segunda é o bloco `trinoCerts`, usado para organizar keystores, certificados e materiais TLS em Secret, PVC e Job de cópia.
A terceira é o controle de acesso com Apache Ranger, configurado por `tdp-trino.accessControl` e arquivos XML adicionais no coordinator.

A ideia prática é simples: LDAP autentica quem entra, HTTPS protege o transporte e Ranger decide o que cada usuário pode acessar.
Quando LDAP ou Ranger estiverem ativos, trate certificados, senhas e shared secrets como dependências obrigatórias do deploy.

Modos de operação

O chart foi desenhado para alternar entre um modo simples de desenvolvimento e um modo autenticado.
Essa diferença é controlada principalmente por `ldap.enabled` e pelos valores de HTTPS/autenticação em `tdp-trino.server.config`.

Modo	`ldap.enabled`	Porta	Autenticação	Certificados
HTTP sem LDAP	`false`	`8080`	Não exige login	Não monta certificados por padrão

© Copyright Tecnisys 2026. All rights reserved.

Modo	`ldap.enabled`	Porta	Autenticação	Certificados
HTTPS com LDAP	`true`	`8443`	`PASSWORD` com LDAP	Monta keystore e configura shared secret

Com `ldap.enabled: false`, o chart usa HTTP na porta 8080 e não exige autenticação. Esse modo é útil para teste controlado, mas não deve ser exposto fora de ambientes restritos.

Com `ldap.enabled: true`, o acesso passa para HTTPS na porta 8443.
Nesse modo, o Trino precisa de keystore, senha do keystore, `internal-communication.shared-secret` e um arquivo `password-authenticator.properties` apontando para o LDAP.

Pré-requisitos

- Servidor LDAP acessível a partir do namespace do Trino, quando LDAP for usado.
- Keystore JKS válido para HTTPS e senha correspondente.
- Secret Kubernetes com os arquivos de certificado antes do deploy.
- StorageClass disponível se `trinoCerts.pvc.enabled: true`.
- Ranger implantado e acessível quando `access-control.name=ranger` for usado.
- Conectividade do coordinator com LDAP, Ranger e fontes de dados protegidas.

Certificados TLS do Trino (`trinoCerts`)

O bloco `trinoCerts` prepara os materiais de certificado usados pelo Trino quando o ambiente exige HTTPS, LDAP ou integrações como Ranger.
Ele não gera certificados automaticamente; ele organiza como arquivos já existentes serão disponibilizados aos pods.

O Secret guarda arquivos sensíveis, como keystores JKS.
O PVC fornece um volume persistente para reutilizar esses arquivos entre etapas do Helm.
O `copyJob` roda como hook de `pre-install` e `pre-upgrade` para copiar arquivos do Secret para o PVC antes da instalação ou atualização.

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Uso	Padrão
`trinoCerts.pvc.enabled`	Cria PVC para certificados	`false`
`trinoCerts.pvc.name`	Nome do PVC	`trino-certs-pvc`
`trinoCerts.pvc.size`	Tamanho do PVC	`1Gi`
`trinoCerts.secret.enabled`	Monta Secret com certificados no Job	`false`
`trinoCerts.secret.name`	Nome do Secret de certificados	`tdp-trino-certs-secret`
`trinoCerts.copyJob.enabled`	Executa Job para preparar o PVC	`false`

Modelo de Secret para LDAP com keystore:

```
kubectl -n <NAMESPACE> create secret generic tdp-trino-certs-secret \
  --from-file=keystore.jks=<LOCAL_PATH>/keystore.jks
```

Exemplo de Secret para fluxo com PVC e Ranger:

```
kubectl -n <NAMESPACE> create secret generic tdp-trino-certs-secret \
  --from-file=trino_cert.jks=<LOCAL_PATH>/trino_cert.jks
```

Exemplo de preparação do PVC por `copyJob`:

```
trinoCerts:
  pvc:
    enabled: true
    size: "1Gi"
  secret:
    enabled: true
    name: "tdp-trino-certs-secret"
```

© Copyright Tecnisys 2026. All rights reserved.

```
  copyJob:
    enabled: true
```

LDAP habilitado

LDAP é usado pelo Trino como autenticador de senha.
No chart, habilitar LDAP implica também habilitar HTTPS, porque credenciais de usuário não devem trafegar por HTTP simples.

O arquivo `password-authenticator.properties` define como o Trino consulta o diretório LDAP.
O `userBindPattern` é o molde do DN usado no bind; `${USER}` é substituído pelo usuário informado no cliente Trino.

Modelo de configuração:

```
ldap:
  enabled: true
  server:
    url: "ldap://<LDAP_HOST>:389"
    insecure: true
  auth:
    userBindPattern: "uid=${USER},cn=users,cn=accounts,dc=
<LDAP_DOMAIN>,dc=com,dc=br"

tdp-trino:
  server:
    config:
      https:
        enabled: true
        port: 8443
        keystore:
          path: "/etc/trino/certs/keystore.jks"
      authenticationType: "PASSWORD"
  additionalConfigProperties:
    - "internal-communication.shared-secret=<SHARED_SECRET>"
    - "http-server.https.keystore.key=<KEYSTORE_PASSWORD>"
  coordinator:
    additionalVolumes:
      - name: trino-certs
        secret:
          secretName: tdp-trino-certs-secret
```

© Copyright Tecnisys 2026. All rights reserved.

```
    additionalVolumeMounts:
      - name: trino-certs
        mountPath: /etc/trino/certs
        readOnly: true
    additionalConfigFiles:
      password-authenticator.properties: |
        password-authenticator.name=ldap
        ldap.url=ldap://<LDAP_HOST>:389
        ldap.allow-insecure=true
        ldap.user-bind-pattern=uid=${USER},cn=users,cn=accounts,dc=
<LDAP_DOMAIN>,dc=com,dc=br
        ldap.cache-ttl=1h
        ldap.timeout.connect=1m
        ldap.timeout.read=1m
```

Substitua os placeholders antes de instalar.
`<LDAP_HOST>` é o endereço do LDAP visto de dentro do cluster.
`<LDAP_DOMAIN>` representa os componentes do DN do diretório.
`<SHARED_SECRET>` deve ser um valor forte para comunicação interna do Trino.
`<KEYSTORE_PASSWORD>` deve corresponder à senha do keystore usado por HTTPS.

Para desabilitar LDAP e voltar ao modo HTTP, remova a configuração LDAP específica e deixe `ldap.enabled: false`.
Depois do upgrade, valide se os pods reiniciaram e se o cliente usa a porta 8080.

Para habilitar LDAP, crie primeiro o Secret de certificados, aplique os valores de LDAP/HTTPS e use a porta 8443 no cliente.
Depois do upgrade, confira o ConfigMap gerado, os mounts do coordinator e os logs de autenticação.

Ranger

Apache Ranger adiciona controle de acesso centralizado ao Trino via plugin de autorização.

A integração usa `tdp-trino.accessControl` para ativar o plugin Ranger.
Os arquivos extras em `tdp-trino.coordinator.additionalConfigFiles` entregam a configuração de política, auditoria e SSL que o plugin lê no startup.

© Copyright Tecnisys 2026. All rights reserved.

Use o exemplo abaixo como modelo de estrutura.
Ele mostra onde ficam os pontos principais sem trazer XML completo para a página.

```
trinoCerts:
  pvc:
    enabled: true
    size: "1Gi"
  secret:
    enabled: true
    name: "tdp-trino-certs-secret"
  copyJob:
    enabled: true

tdp-trino:
  server:
    config:
      https:
        enabled: true
        port: 8443
        keystore:
          path: "/etc/trino/ssl/trino_cert.jks"
          key: "<KEYSTORE_PASSWORD>"
      authenticationType: "PASSWORD"

  accessControl:
    type: properties
    properties: |
      access-control.name=ranger
      ranger.service.name=dev_trino
      ranger.plugin.config.resource=/etc/trino/ranger-trino-
security.xml,/etc/trino/ranger-trino-audit.xml,/etc/trino/ranger-policymgr-
ssl.xml

  coordinator:
    additionalVolumes:
      - name: trino-certs
        persistentVolumeClaim:
          claimName: trino-certs-pvc
    additionalVolumeMounts:
      - name: trino-certs
        mountPath: /etc/trino/ssl
    additionalConfigFiles:
      ranger-trino-security.xml: |
        <?xml version="1.0" encoding="UTF-8"?>
```

© Copyright Tecnisys 2026. All rights reserved.

```
    <configuration>
      <property>
        <name>ranger.plugin.trino.policy.rest.url</name>
        <value>http://ranger-<NAMESPACE>.svc.cluster.local:6180</value>
      </property>
      <property>
        <name>ranger.plugin.trino.super.users</name>
        <value>admin</value>
      </property>
    </configuration>
  ranger-trino-audit.xml: |
    <configuration>
      <!-- Configuração de auditoria do plugin -->
    </configuration>
  ranger-policymgr-ssl.xml: |
    <configuration>
      <!-- Truststore/keystore para comunicação com Ranger -->
    </configuration>
```

O valor `ranger.service.name` deve ser o mesmo serviço cadastrado na interface do Ranger.
Se os nomes não baterem, o plugin pode iniciar, mas as políticas não serão aplicadas ao serviço esperado.

Use superusuários com cuidado.
Uma conta listada como superusuário passa pelas políticas com privilégio amplo, então mantenha a lista curta e auditável.

Quando houver auditoria externa ou SSL mútuo, preencha os XMLs com os destinos, truststores e credenciais exigidos pelo ambiente.
Mantenha senhas e arquivos sensíveis fora do repositório.

Conectar com cliente Trino

Modelo para HTTP sem LDAP:

HTTP sem LDAP

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME> 8080:8080 &
trino --server http://localhost:8080
```

© Copyright Tecnisys 2026. All rights reserved.

Modelo para HTTPS com LDAP:

HTTPS com LDAP

```
kubectl -n <NAMESPACE> port-forward svc/<RELEASE_NAME> 8443:8443 &
trino --server https://localhost:8443 \
      --user <LDAP_USER> \
      --password \
      --insecure
```

Exemplo:

HTTP sem LDAP

```
kubectl port-forward -n tdp-project svc/tdp-trino 8080:8080 &
trino --server http://localhost:8080
```

HTTPS com LDAP

```
kubectl port-forward -n tdp-project svc/tdp-trino 8443:8443 &
trino --server https://localhost:8443 \
      --user your-ldap-username \
      --password \
      --insecure
```

Em produção, prefira validar a cadeia de certificado em vez de usar `--insecure`.
Use `--insecure` apenas para depuração controlada ou certificados de laboratório.

Parâmetros principais

Parâmetro	Papel	Padrão	Quando alterar
`ldap.enabled`	Liga ou desliga o modo LDAP	`false`	Ambientes com autenticação corporativa

© Copyright Tecnisys 2026. All rights reserved.

Parâmetro	Papel	Padrão	Quando alterar
`tdp-trino.server.config.https.enabled`	Habilita HTTPS no Trino	`false`	LDAP, Ranger ou exposição segura
`tdp-trino.server.config.https.port`	Porta HTTPS do Trino	`8443`	Quando o ambiente padronizar outra porta
`tdp-trino.server.config.https.keystore.path`	Caminho do keystore dentro do pod	vazio	Sempre que HTTPS for usado
`tdp-trino.server.config.authenticationType`	Tipo de autenticação do Trino	vazio	Use `PASSWORD` para LDAP ou arquivo de senhas
`tdp-trino.additionalConfigProperties`	Propriedades extras do Trino	`[]`	Shared secret, HTTPS interno e ajustes avançados
`tdp-trino.coordinator.additionalVolumes`	Volumes extras no coordinator	`[]`	Montar Secret ou PVC com certificados
`tdp-trino.coordinator.additionalConfigFiles`	Arquivos extras no coordinator	`{}`	LDAP, password file e XMLs do Ranger
`tdp-trino.accessControl`	Configuração de autorização	`{}`	Apache Ranger ou outro controle suportado
`trinoCerts.*`	Preparação de certificados	desabilitado	HTTPS, LDAP, Ranger ou SSL mútuo

© Copyright Tecnisys 2026. All rights reserved.

Solução de problemas

Situação	O que verificar
Pod não sobe com LDAP	Caminho do keystore, senha do keystore, Secret, mounts e `authenticationType`
Login LDAP falha	URL LDAP, `userBindPattern`, conectividade, certificado e permissões do usuário de bind
Cliente não conecta	Porta correta: 8080 para HTTP, 8443 para HTTPS
Erro de certificado	Nome do arquivo no Secret, mountPath e path configurado no Trino
Ranger não aplica políticas	`ranger.service.name`, URL do Ranger, XMLs no coordinator e usuário autenticado
PVC não é criado	`trinoCerts.pvc.enabled`, StorageClass e eventos do namespace

Ao trocar de LDAP para HTTP, remova os blocos específicos de LDAP/HTTPS para evitar configuração residual.
Depois valide se o Service e o cliente voltaram para a porta 8080.

Ao trocar de HTTP para LDAP, crie o Secret antes do `helm upgrade`.
Depois valide o ConfigMap, os volumes montados e o acesso pela porta 8443.

Boas práticas

- Use LDAPS na porta 636 em produção sempre que o diretório suportar.
- Não versionar keystores, senhas, hashes reais ou shared secrets.
- Rotacione senhas de keystore e shared secrets periodicamente.
- Restrinja permissões do usuário de bind LDAP.

© Copyright Tecnisys 2026. All rights reserved.

- Mantenha a lista de superusuários do Ranger mínima.
- Monitore logs de autenticação do Trino e auditoria do Ranger.
- Deixe `ldap.enabled: false` apenas em ambientes controlados de desenvolvimento ou teste.

© Copyright Tecnisys 2026. All rights reserved.

Trino – Integração

CHART

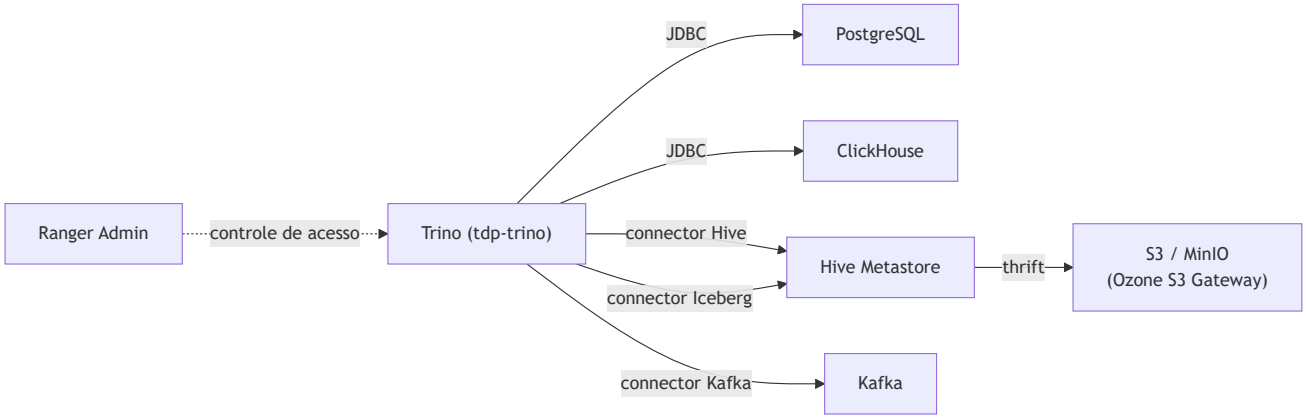
Version 3.0.1 | Type application | AppVersion 478

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+

Visão geral

O chart `tdp-trino` integra fontes de dados por meio de **catálogos** definidos em `tdp-trino.catalogs`.
Cada catálogo é um bloco de propriedades do connector Trino e fica disponível para consultas no formato `catalog.schema.objeto`.



Os exemplos abaixo cobrem ClickHouse, Hive/Iceberg com S3, Kafka e PostgreSQL. Use-os como ponto de partida e ajuste hosts, portas, Secrets e propriedades conforme o ambiente.

Padrão de configuração

Exemplo

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-trino -n <NAMESPACE> -f
values.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

Defina os catálogos sob `tdp-trino.catalogs`:

```
tdp-trino:
  catalogs:
    <nome-do-catálogo>: |
      connector.name=<CONNECTOR_NAME>
      propriedade=valor
```

Catálogos padrão

Catálogo	Uso
`tpch`	Disponibiliza dados sintéticos do benchmark TPC-H, úteis para testes e validações iniciais.
`tpcds`	Disponibiliza dados sintéticos do benchmark TPC-DS, voltados a cenários analíticos/data warehouse.
`memory`	Permite criar tabelas temporárias mantidas em memória, útil para testes simples.

PostgreSQL

```
tdp-trino:
  catalogs:
    postgresql: |
      connector.name=postgresql
      connection-url=jdbc:postgresql://<POSTGRESQL_SERVICE>:5432/<DATABASE_NAME>
      connection-user=<USERNAME>
      connection-password=<PASSWORD>
```

Pré-requisitos: PostgreSQL acessível a partir do namespace do Trino, usuário com permissões coerentes com as consultas esperadas e credenciais protegidas por Secret ou mecanismo externo quando necessário.

© Copyright Tecnisys 2026. All rights reserved.

ClickHouse

```
tdp-trino:
  catalogs:
    clickhouse: |
      connector.name=clickhouse
      connection-url=jdbc:clickhouse://<CLICKHOUSE_SERVICE>:8123/<DATABASE_NAME>
      connection-user=<USERNAME>
      connection-password=<PASSWORD>
      domain-compaction-threshold=1000
```

Pré-requisitos: ClickHouse implantado e acessível pela rede do cluster, porta HTTP/JDBC liberada para o Trino e Secret para senha quando a credencial não puder ficar no arquivo de values.

Exemplo para criar um Secret de senha, quando necessário:

```
kubectl create secret generic tdp-clickhouse-trino-secret \
  --from-literal=password=<PASSWORD> \
  --namespace <NAMESPACE>
```

Hive e Iceberg com S3/Ozone

Use Hive Metastore como catálogo de metadados e um endpoint S3 compatível, como Ozone S3 Gateway ou MinIO.

```
tdp-trino:
  catalogs:
    hive: |
      connector.name=hive
      hive.metastore.uri=thrift://<HIVE_METASTORE_SERVICE>:9083
      hive.non-managed-table-writes-enabled=true
      hive.non-managed-table-creates-enabled=true
      fs.native-s3.enabled=true
      s3.endpoint=<S3_ENDPOINT>
      s3.region=us-east-1
      s3.aws-access-key=<S3_ACCESS_KEY>
      s3.aws-secret-key=<SECRET_KEY>
      s3.path-style-access=true
    iceberg: |
```

© Copyright Tecnisys 2026. All rights reserved.

```
      connector.name=iceberg
      iceberg.catalog.type=hive_metastore
      hive.metastore.uri=thrift://<HIVE_METASTORE_SERVICE>:9083
      fs.native-s3.enabled=true
      s3.endpoint=<S3_ENDPOINT>
      s3.region=us-east-1
      s3.aws-access-key=<S3_ACCESS_KEY>
      s3.aws-secret-key=<SECRET_KEY>
      s3.path-style-access=true
```

Pré-requisitos: Hive Metastore acessível na porta Thrift, storage S3 compatível acessível pelo Trino e credenciais de acesso ao bucket/volume.

Kafka

```
tdp-trino:
  catalogs:
    kafka: |
      connector.name=kafka
      kafka.nodes=<KAFKA_BOOTSTRAP_SERVICE>:9092
      kafka.table-description-dir=/etc/trino/kafka
      kafka.default-schema=default
      kafka.hide-internal-columns=false
      kafka.buffer-size=64kB
```

Pré-requisitos: brokers Kafka acessíveis a partir do Trino, descrições de tabelas do connector Kafka quando necessário e Schema Registry quando usar formatos como Avro ou Protobuf.

Ranger e controle de acesso

Integração com Ranger não é definida em `tdp-trino.catalogs`.
Ela usa `tdp-trino.accessControl`, arquivos adicionais do coordinator e, em muitos cenários, certificados.
Consulte Segurança - Trino.

Validação

```
kubectl -n <NAMESPACE> get pods
kubectl -n <NAMESPACE> get configmap
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl -n <NAMESPACE> logs deploy/<DEPLOYMENT_NAME>
```

```
trino --server http://<TRINO_SERVICE>:8080 --user <USERNAME> --catalog tpch --
schema tiny
```

Para HTTPS/LDAP, use as instruções de Segurança - Trino.

© Copyright Tecnisys 2026. All rights reserved.

ATUALIZAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Atualização via Helm

COMPATIBILIDADE

Kubernetes	OpenShift	Rancher	Helm
1.32+	4.19+	2.10.x+	3.2.0+

Este guia descreve o processo de atualização dos componentes do TDP Kubernetes utilizando o **Helm CLI**.
A atualização via Helm permite aplicar novas versões dos charts, atualizar configurações e garantir que o ambiente esteja sempre alinhado com as versões mais recentes.

Preparo

Preparo da Atualização

Antes de iniciar qualquer atualização, execute as verificações abaixo para garantir um processo seguro.

Comandos

Verificar Versões Atuais

Liste todos os releases instalados no namespace para identificar as versões atuais:

Terminal input

```
helm list -n <NAMESPACE>
```

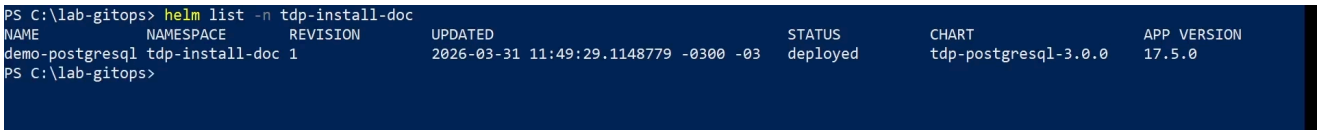


Figura 1 - Helm Releases

A saída exibirá nome do release, namespace, revisão, status, chart e versão da aplicação.

© Copyright Tecnisys 2026. All rights reserved.

Verificar os pods e a prontidão dos workloads

Verifique se os workloads estão prontos no namespace de destino antes de prosseguir:

Terminal input

```
kubectl get pods -n <NAMESPACE>
kubectl get statefulset -n <NAMESPACE>
```



Figura 2 - Get Pods



Figura 3 - Get Statefulset

Se a aplicação utilizar Deployment, valide também:

Terminal input

```
kubectl get deploy -n <NAMESPACE>
```

O que validar:

- Os pods estão com `STATUS` **Running**
- A coluna `READY` dos pods está completa, por exemplo `1/1`
- Os StatefulSets ou Deployments da aplicação mostram todas as réplicas prontas, por exemplo `1/1`, `2/2` ou `3/3`

Por que validar antes da atualização:

- Garantir que a aplicação já está estável antes da mudança
- Evitar confundir falha anterior com falha causada pela atualização
- Reduzir o risco de indisponibilidade durante o processo

Backup dos Valores Atuais

© Copyright Tecnisys 2026. All rights reserved.

Exporte os valores atuais de cada release para possibilitar rollback em caso de falha:

Terminal input

```
helm get values <RELEASE_NAME> -n <NAMESPACE> -o yaml > <RELEASE_NAME>-values-backup.yaml
```

Repita este comando para cada release que será atualizado.

Backup dos Volumes Persistentes

> **CAUTION**
>
> Antes de qualquer atualização, realize backup dos dados persistentes. Em caso de falha grave, esses backups serão necessários para recuperação.

1. Identifique os PVCs do componente:

Terminal input

```
kubectl get pvc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

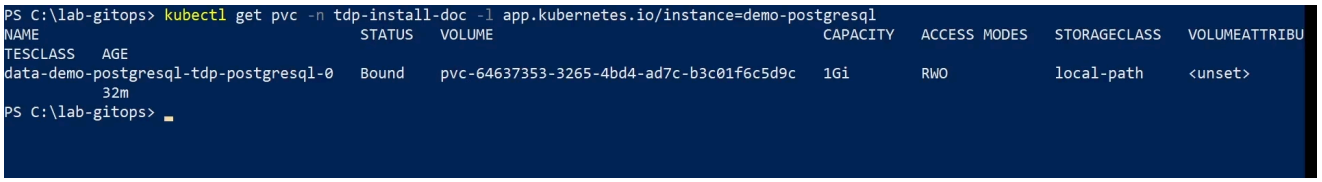


Figura 4 - Get Pvc

2. Crie snapshots dos volumes persistentes (se o StorageClass suportar):

Terminal input

```
kubectl apply -f - <<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: <RELEASE_NAME>-snapshot
  namespace: <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
spec:
  volumeSnapshotClassName: <snapshot-class>
  source:
    persistentVolumeClaimName: <pvc-name>
EOF
```

3. Verifique o status do snapshot:

Terminal input

```
kubectl get volumesnapshot -n <NAMESPACE>
```

Atualização dos CRDs

> **WARNING**
>
> Os CRDs devem ser atualizados **antes** dos demais componentes. São recursos de escopo global no cluster e não são gerenciados automaticamente pelo Helm durante o upgrade.

Atualize os CRDs da plataforma (`tdp-crds`) antes de qualquer outro componente:

Terminal input

```
helm upgrade --install <RELEASE_NAME> \
  oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
  -n <NAMESPACE> --create-namespace
```

> **EXEMPLO**
>
> Exemplo
>
> ```
> helm upgrade --install tdp-crds \
>   oci://registry.tecnisys.com.br/tdp/charts/tdp-crds \
>   -n <NAMESPACE> --create-namespace
> ```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> helm upgrade --install tdp-crds oci://registry.tecnisys.com.br/tdp/charts/tdp-crds -n tdp-install-doc --create-namespace
Pulled: registry.tecnisys.com.br/tdp/charts/tdp-crds:3.0.0
Digest: sha256:b507c3dd295541bdb915057319d52e274f05ba352c11643c1f92b157f51a7016
Release "tdp-crds" has been upgraded. Happy Helming!
NAME: tdp-crds
LAST DEPLOYED: Thu Apr 23 09:35:21 2026
NAMESPACE: tdp-install-doc
STATUS: deployed
REVISION: 3
TEST SUITE: None
PS C:\lab-gitops>
```

Figura 5 - Helm Upgrade tdp-crds

Aguarde a conclusão e verifique se os CRDs foram atualizados:

Terminal input

```
kubectl get crds | grep argoproj.io
```

```
PS C:\lab-gitops> kubectl get crds
NAME                                                  CREATED AT
accesscontrolpolicies.hub.traefik.io                  2026-03-13T12:04:24Z
addons.k3s.cattle.io                                  2026-03-13T12:03:51Z
aiservices.hub.traefik.io                             2026-03-13T12:04:24Z
apiaccesses.hub.traefik.io                            2026-03-13T12:04:24Z
apibundles.hub.traefik.io                             2026-03-13T12:04:24Z
apicatalogitems.hub.traefik.io                        2026-03-13T12:04:24Z
apiplans.hub.traefik.io                               2026-03-13T12:04:24Z
apiportals.hub.traefik.io                             2026-03-13T12:04:24Z
apiratelimits.hub.traefik.io                          2026-03-13T12:04:24Z
apis.hub.traefik.io                                   2026-03-13T12:04:24Z
apiversions.hub.traefik.io                            2026-03-13T12:04:24Z
applications.argoproj.io                              2026-03-19T15:19:20Z
applicationsets.argoproj.io                           2026-03-19T15:19:21Z
appprojects.argoproj.io                               2026-03-19T15:19:21Z
certificaterequests.cert-manager.io                   2026-04-23T12:12:48Z
certificates.cert-manager.io                          2026-04-23T12:12:48Z
challenges.acme.cert-manager.io                       2026-04-23T12:12:48Z
clickhouseinstallations.clickhouse.altinity.com       2026-04-23T12:12:45Z
clickhouseinstallationtemplates.clickhouse.altinity.com 2026-04-23T12:12:45Z
clickhousekeeperinstallations.clickhouse-keeper.altinity.com 2026-04-23T12:12:45Z
clickhouseoperatorconfigurations.clickhouse.altinity.com 2026-04-23T12:12:46Z
clusterissuers.cert-manager.io                        2026-04-23T12:12:48Z
cronomjobs.pipelines.openmetadata.org                 2026-04-23T12:12:51Z
etcdsnapshotfiles.k3s.cattle.io                       2026-03-13T12:03:51Z
gatewayclasses.gateway.networking.k8s.io              2026-03-13T12:04:24Z
gateways.gateway.networking.k8s.io                    2026-03-13T12:04:24Z
grpcroutes.gateway.networking.k8s.io                  2026-03-13T12:04:24Z
helmchartconfigs.helm.cattle.io                       2026-03-13T12:03:51Z
helmcharts.helm.cattle.io                             2026-03-13T12:03:51Z
httproutes.gateway.networking.k8s.io                  2026-03-13T12:04:24Z
ingressroutes.traefik.io                              2026-03-13T12:04:24Z
ingressroutetcps.traefik.io                           2026-03-13T12:04:24Z
ingressrouteudps.traefik.io                           2026-03-13T12:04:24Z
issuers.cert-manager.io                               2026-04-23T12:12:48Z
kafkabridges.kafka.strimzi.io                         2026-04-14T11:17:00Z
```

Figura 6 - Verifique os cdrs

Verifique o histórico:

Terminal input

```
helm history <RELEASE_NAME> -n <NAMESPACE>
```

EXEMPLO

© Copyright Tecnisys 2026. All rights reserved.

Comando usado nas capturas desta documentação:

Exemplo

```
helm history demo-postgresql -n tdp-install-doc
```



Figura 7 - Verifica o Histórico

Atualização

Atualização de Componentes

Comandos

Comando de Atualização

O comando padrão para atualizar um componente do TDP Kubernetes é:

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  -n <NAMESPACE> \
  -f values.yaml
```

Onde:

- `<RELEASE_NAME>`: nome do release Helm (ex.: `tdp-kafka`, `tdp-trino`)
- `<chart>`: nome do chart no registry (ex.: `tdp-kafka`, `tdp-trino`)
- `values.yaml`: arquivo com as configurações personalizadas do componente

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> helm upgrade demo-postgresql oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql -n tdp-install-doc -f .\values-postgresql.
yaml
Pulled: registry.tecnisys.com.br/tdp/charts/tdp-postgresql:3.0.0
Digest: sha256:accc1eab0a3199e335f43b335fd8d8d79765c1f08ecafd7920b1a803b4fa0a6e
Release "demo-postgresql" has been upgraded. Happy Helming!
NAME: demo-postgresql
LAST DEPLOYED: Tue Mar 31 12:33:06 2026
NAMESPACE: tdp-install-doc
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
PostgreSQL has been deployed successfully.

## Access from inside the cluster

The PostgreSQL service can be accessed on port 5432 using:
  demo-postgresql.tdp-install-doc.svc.cluster.local

## Get the postgres password

export POSTGRES_PASSWORD=$(kubectl get secret --namespace tdp-install-doc demo-postgresql -o jsonpath="{.data['postgres-password']}" | base64 -d)

## Connect and test (psql)

kubectl run demo-postgresql-client --rm --tty -i --restart='Never' --namespace tdp-install-doc \
  --image registry.tecnisys.com.br/community/images/bitnamilegacy/postgresql:17.5.0-debian-12-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
  --command -- psql -h demo-postgresql -p 5432 -U postgres -d postgres

## Access from outside the cluster (port-forward)

kubectl port-forward --namespace tdp-install-doc svc/demo-postgresql 5432:5432 &
psql -h 127.0.0.1 -p 5432 -U postgres -d postgres
PS C:\lab-gitops>
```

Figura 7 - Helm Upgrade - Comando padrão (exemplo: crds)

Ordem de Atualização Recomendada

> **COMPATIBILIDADE DE VERSÕES**
>
> Antes de atualizar, consulte as Notas de Release do TDP para verificar compatibilidade entre versões de charts. Em caso de dúvida, entre em contato com o suporte Tecnisys.

Respeite a ordem abaixo para garantir que as dependências sejam atendidas. Em cada comando, substitua `<RELEASE_NAME>` pelo nome do release Helm **daquele** chart (muitas equipes usam o prefixo `tdp-` igual ao nome do chart).

1. **CRDs e Infraestrutura**

Terminal input

```
helm upgrade --install <RELEASE_NAME>
oci://registry.tecnisys.com.br/tdp/charts/tdp-crds -n <NAMESPACE> --create-
namespace
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-argo -
n <NAMESPACE> -f values-argo.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

EXEMPLO

Exemplo

```
helm upgrade --install tdp-crds
oci://registry.tecnisys.com.br/tdp/charts/tdp-crds -n <NAMESPACE> --
create-namespace
helm upgrade tdp-argo oci://registry.tecnisys.com.br/tdp/charts/tdp-
argo -n <NAMESPACE> -f values-argo.yaml
```

2. **Banco de Dados**

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
postgresql -n <NAMESPACE> -f values-postgresql.yaml
```

EXEMPLO

Exemplo

```
helm upgrade tdp-postgresql
oci://registry.tecnisys.com.br/tdp/charts/tdp-postgresql -n <NAMESPACE>
-f values-postgresql.yaml
```

3. **Mensageria**

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka
-n <NAMESPACE> -f values-kafka.yaml
```

EXEMPLO

© Copyright Tecnisys 2026. All rights reserved.

Exemplo

```
helm upgrade tdp-kafka oci://registry.tecnisys.com.br/tdp/charts/tdp-kafka -n <NAMESPACE> -f values-kafka.yaml
```

4. Camada de Armazenamento e Metadados

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore -n <NAMESPACE> -f values-hive-metastore.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg -n <NAMESPACE> -f values-iceberg.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone -n <NAMESPACE> -f values-ozone.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake -n <NAMESPACE> -f values-deltalake.yaml
```

EXEMPLO

Exemplo

```
helm upgrade tdp-hive-metastore oci://registry.tecnisys.com.br/tdp/charts/tdp-hive-metastore -n <NAMESPACE> -f values-hive-metastore.yaml
helm upgrade tdp-iceberg oci://registry.tecnisys.com.br/tdp/charts/tdp-iceberg -n <NAMESPACE> -f values-iceberg.yaml
helm upgrade tdp-ozone oci://registry.tecnisys.com.br/tdp/charts/tdp-ozone -n <NAMESPACE> -f values-ozone.yaml
helm upgrade tdp-deltalake oci://registry.tecnisys.com.br/tdp/charts/tdp-deltalake -n <NAMESPACE> -f values-deltalake.yaml
```

5. Segurança e Governança

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-ranger
-n <NAMESPACE> -f values-ranger.yaml
```

EXEMPLO

Exemplo

```
helm upgrade tdp-ranger oci://registry.tecnisys.com.br/tdp/charts/tdp-
ranger -n <NAMESPACE> -f values-ranger.yaml
```

6. **Processamento e Ingestão**

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-nifi -
n <NAMESPACE> -f values-nifi.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-spark
-n <NAMESPACE> -f values-spark.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-trino
-n <NAMESPACE> -f values-trino.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-
clickhouse -n <NAMESPACE> -f values-clickhouse.yaml
```

EXEMPLO

Exemplo

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade tdp-nifi oci://registry.tecnisys.com.br/tdp/charts/tdp-nifi -n <NAMESPACE> -f values-nifi.yaml
helm upgrade tdp-spark oci://registry.tecnisys.com.br/tdp/charts/tdp-spark -n <NAMESPACE> -f values-spark.yaml
helm upgrade tdp-trino oci://registry.tecnisys.com.br/tdp/charts/tdp-trino -n <NAMESPACE> -f values-trino.yaml
helm upgrade tdp-clickhouse oci://registry.tecnisys.com.br/tdp/charts/tdp-clickhouse -n <NAMESPACE> -f values-clickhouse.yaml
```

7. Ferramentas e Interfaces

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow -n <NAMESPACE> -f values-airflow.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter -n <NAMESPACE> -f values-jupyter.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-superset -n <NAMESPACE> -f values-superset.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata -n <NAMESPACE> -f values-openmetadata.yaml
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver -n <NAMESPACE> -f values-cloudbeaver.yaml
```

EXEMPLO

Exemplo

```
helm upgrade tdp-airflow oci://registry.tecnisys.com.br/tdp/charts/tdp-airflow -n <NAMESPACE> -f values-airflow.yaml
helm upgrade tdp-jupyter oci://registry.tecnisys.com.br/tdp/charts/tdp-jupyter -n <NAMESPACE> -f values-jupyter.yaml
helm upgrade tdp-superset oci://registry.tecnisys.com.br/tdp/charts/tdp-superset -n <NAMESPACE> -f values-superset.yaml
helm upgrade tdp-openmetadata oci://registry.tecnisys.com.br/tdp/charts/tdp-openmetadata -n <NAMESPACE> -f values-openmetadata.yaml
```

© Copyright Tecnisys 2026. All rights reserved.

```
helm upgrade tdp-cloudbeaver
oci://registry.tecnisys.com.br/tdp/charts/tdp-cloudbeaver -n
<NAMESPACE> -f values-cloudbeaver.yaml
```

Atualização com Valores Personalizados

Para atualizar um componente aplicando valores adicionais, utilize `--set` em conjunto com o arquivo de valores:

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  -n <NAMESPACE> \
  -f values.yaml \
  --set replicaCount=3 \
  --set resources.requests.memory=4Gi
```

Ou combine múltiplos arquivos de valores:

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  -n <NAMESPACE> \
  -f values.yaml \
  -f values-production.yaml
```

> **NOTE**
>
> Quando múltiplos arquivos de valores são fornecidos, o último arquivo tem precedência sobre os anteriores.

Verificar a Versão Disponível

Antes de atualizar, verifique a versão mais recente disponível no registry:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
helm show chart oci://registry.tecnisys.com.br/tdp/charts/<chart>
```

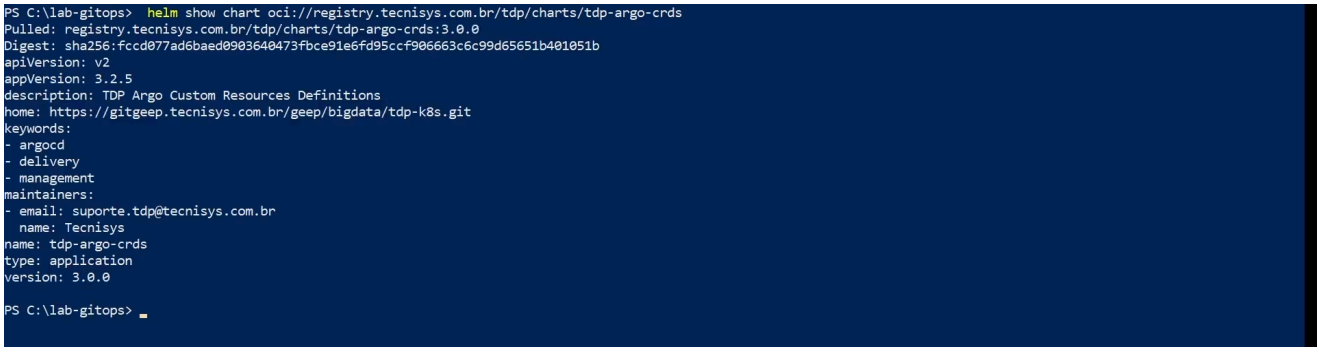


Figura 9 - show-chart - exemplo

Para atualizar para uma versão específica:

Terminal input

```
helm upgrade <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  -n <NAMESPACE> \
  -f values.yaml \
  --version <CHART_VERSION>
```

Compatibilidade de Versões

Ao atualizar componentes do TDP Kubernetes, observe:

- **Atualizações incrementais**: sempre atualize versão a versão sequencialmente. Pular versões intermediárias pode causar incompatibilidades.
- **Dependências entre componentes**: alguns componentes dependem de versões específicas de outros. Por exemplo, o `tdp-trino` pode depender de uma versão mínima do `tdp-hive-metastore`.
- **Compatibilidade com Kubernetes**: verifique se a versão do chart é compatível com a versão do seu cluster Kubernetes.

Validação

Validação Pós-Atualização

Após cada atualização, execute as validações abaixo.

© Copyright Tecnisys 2026. All rights reserved.

Comandos

Verificar os pods e a prontidão dos workloads

Terminal input

```
kubectl get pods -n <NAMESPACE>
kubectl get statefulset -n <NAMESPACE>
```



Figura 10 - Verificar os Pods



Figura 11 - Verificar StatefulSet

Se a aplicação utilizar Deployment, valide também:

Terminal input

```
kubectl get deploy -n <NAMESPACE>
```

O que confirmar:

- Os pods estão com `STATUS` **Running**
- A coluna `READY` dos pods está completa, por exemplo `1/1`
- Os StatefulSets ou Deployments da aplicação mostram todas as réplicas prontas, por exemplo `1/1`, `2/2` ou `3/3`

Verificar os Releases do Helm

Terminal input

```
helm list -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

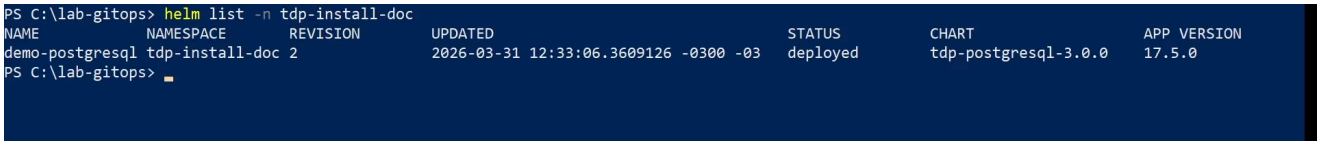


Figura 12 - Verificar os Releases do Helm

Todos os releases devem apresentar o status `deployed`.

Verificar os Serviços

Terminal input

```
kubectl get svc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

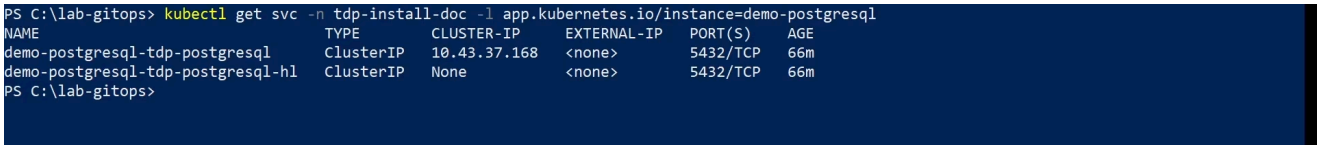


Figura 13 - Verificar os Serviços

Verificar os Volumes Persistentes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```



Figura 14 - Verificar os Volumes Persistentes

Todos os *Persistent Volume Claims* devem apresentar o status `Bound`.

Verificar os Logs

Analise os logs do componente atualizado para identificar possíveis erros:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME> --
tail=100
```

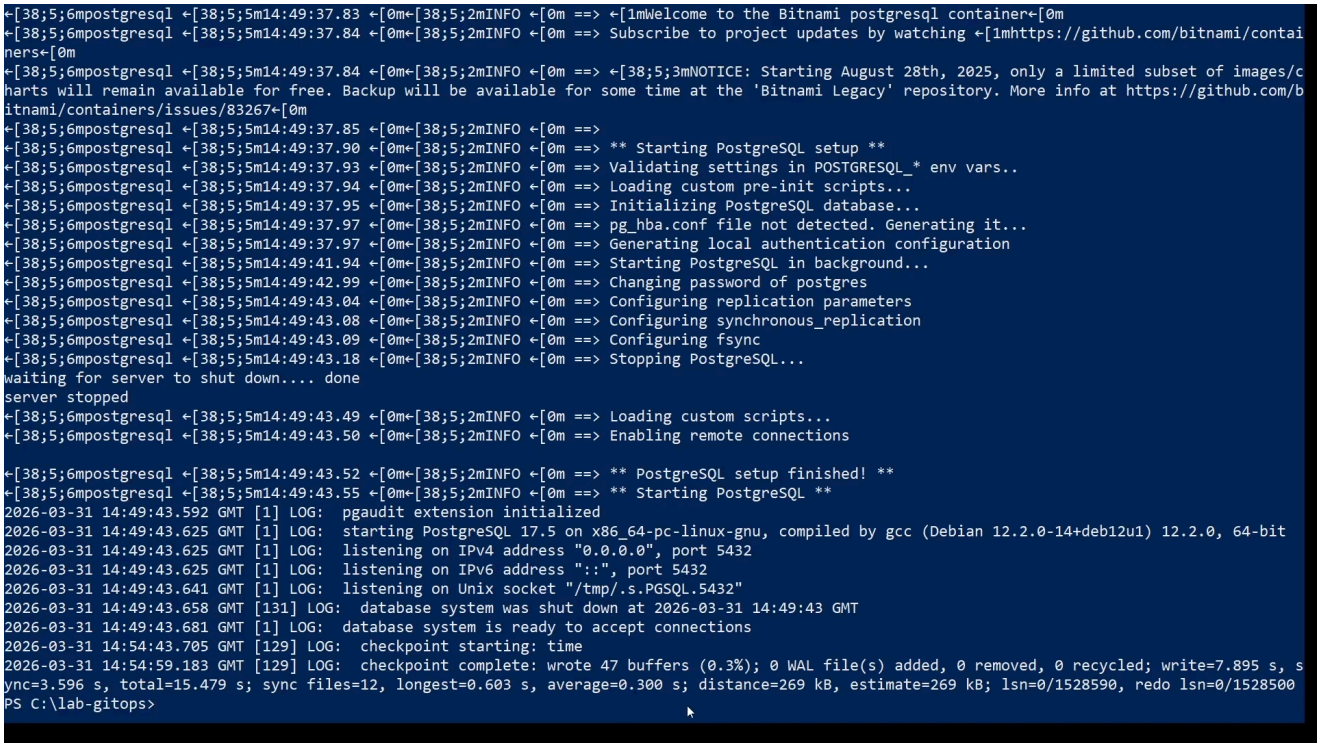


Figura 15 - Verificar os Logs

Verificar a Revisão do Helm

Confirme que a revisão do release foi incrementada com sucesso:

Terminal input

```
helm history <RELEASE_NAME> -n <NAMESPACE>
```

Resolução de Problemas

Resolução de Problemas

Comandos

Identificar o Problema

Verifique o status do release:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm status <RELEASE_NAME> -n <NAMESPACE>
```

Analise os eventos do Kubernetes para identificar a causa da falha:

Terminal input

```
kubectl describe pods -n <NAMESPACE> -l app.kubernetes.io/instance=
<RELEASE_NAME>
kubectl get events -n <NAMESPACE> --sort-by='.lastTimestamp'
```

Reverter a Atualização

Se necessário, realize o rollback para a versão anterior:

Terminal input

```
helm rollback <RELEASE_NAME> -n <NAMESPACE>
```

> **TIP**
>
> Consulte a página de Rollback para instruções detalhadas.

Forçar a Recriação dos Pods

Em alguns casos, pode ser necessário forçar a recriação dos pods para aplicar configurações corretamente:

Terminal input

```
kubectl rollout restart deployment/<deployment> -n <NAMESPACE>
# ou para StatefulSets:
kubectl rollout restart statefulset/<statefulset> -n <NAMESPACE>
```

Para encontrar o nome do Deployment ou StatefulSet:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl get deployments,statefulsets -n <NAMESPACE> -l
app.kubernetes.io/instance=<RELEASE_NAME>
```

> **WARNING**
>
> O rollout restart reinicia os pods de forma controlada (rolling update). Para componentes sem réplicas configuradas, haverá breve indisponibilidade durante a reinicialização.

Desinstalar um Componente com Problema

Para desinstalar e reinstalar um componente específico:

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
helm install <RELEASE_NAME> oci://registry.tecnisys.com.br/tdp/charts/<chart> \
  -n <NAMESPACE> \
  -f values.yaml
```

> **WARNING**
>
> A desinstalação de um release Helm **não remove** os *Persistent Volume Claims* (PVCs) associados. Para uma reinstalação limpa, exclua os PVCs manualmente antes de reinstalar:
>
> Terminal input
>
> ```
> kubectl delete pvc <pvc-name> -n <NAMESPACE>
> ```

© Copyright Tecnisys 2026. All rights reserved.

Atualização via ArgoCD

COMPATIBILIDADE

Kubernetes 1.32+ | OpenShift 4.19+ | Rancher 2.10.x+ | Helm 3.2.0+

Este guia descreve o processo de atualização dos componentes do TDP Kubernetes utilizando o **ArgoCD** como ferramenta de GitOps.
Com o ArgoCD, as atualizações são gerenciadas de forma declarativa por meio dos manifests de Application armazenados no repositório Git.

> **TIP**
>
> Com o padrão GitOps, a atualização consiste em **editar os manifests no repositório Git** — o ArgoCD detecta a mudança e aplica automaticamente ao cluster.

Preparo

Preparo da Atualização

Antes de alterar os manifests, verifique o estado atual do cluster.

Comandos

Verificar o estado das Applications

Liste todas as Applications do TDP e seus status:

Terminal input

```
argocd app list
```

© Copyright Tecnisys 2026. All rights reserved.

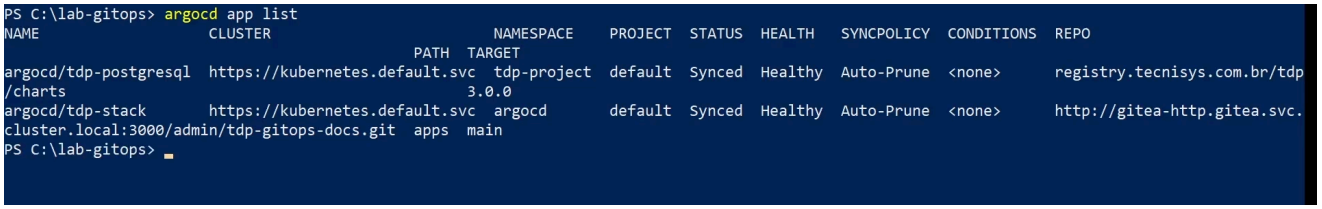


Figura 1 - Applications do TDP e seus status

O que validar:

- Todas as Applications devem estar com `STATUS` **Synced**
- Todas devem estar com `HEALTH` **Healthy**

Por que validar antes da atualização:

- Confirmar que o estado atual no cluster corresponde ao estado desejado no Git
- Garantir que não há erro de reconciliação pendente
- Evitar iniciar a atualização sobre uma Application já inconsistente ou degradada

Verificar os pods e a prontidão dos workloads

Terminal input

```
kubectl get pods -n <NAMESPACE>
```



Figura 2 - Verificar os pods

Terminal input

```
kubectl get statefulset -n <NAMESPACE>
```



Figura 3 - Verificar os StatefulSets

© Copyright Tecnisys 2026. All rights reserved.

Se a aplicação utilizar Deployment, valide também:

Terminal input

```
kubectl get deploy -n <NAMESPACE>
```

O que validar:

- Os pods estão com `STATUS` **Running**
- A coluna `READY` dos pods está completa, por exemplo `1/1`
- Os StatefulSets ou Deployments da aplicação mostram todas as réplicas prontas, por exemplo `1/1`, `2/2` ou `3/3`

Por que validar antes da atualização:

- Garantir que a aplicação já está estável antes da mudança
- Evitar confundir falha anterior com falha causada pela atualização
- Reduzir o risco de indisponibilidade durante o processo

Verificar os Releases do Helm

Terminal input

```
helm list -n <NAMESPACE>
```

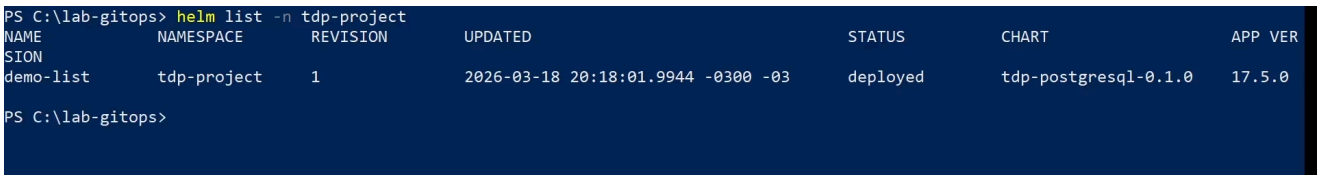


Figura 4 - Verificar os Releases do Helm

Confirme as versões instaladas e o status `deployed` de cada release.

Atualização

Atualização dos Manifests de Application

© Copyright Tecnisys 2026. All rights reserved.



A atualização via ArgoCD consiste em modificar os manifests de `Application` no repositório Git, alterando a versão do chart (`targetRevision`) para a nova versão desejada.

Comandos

Alterar a Versão do Chart

Edite o manifest da Application do componente que deseja atualizar. O campo `targetRevision` define a versão do Helm Chart utilizada. Exemplo para o **PostgreSQL** (componente base que costuma ser instalado primeiro):

apps/tdp-postgresql.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: tdp-postgresql
  namespace: argocd
spec:
  project: default
  source:
    repoURL: oci://registry.tecnisys.com.br/tdp
    chart: tdp-postgresql
    targetRevision: 3.0.1  # Altere para a nova versão desejada
    helm:
      valueFiles:
        - values.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: <NAMESPACE>
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

© Copyright Tecnisys 2026. All rights reserved.

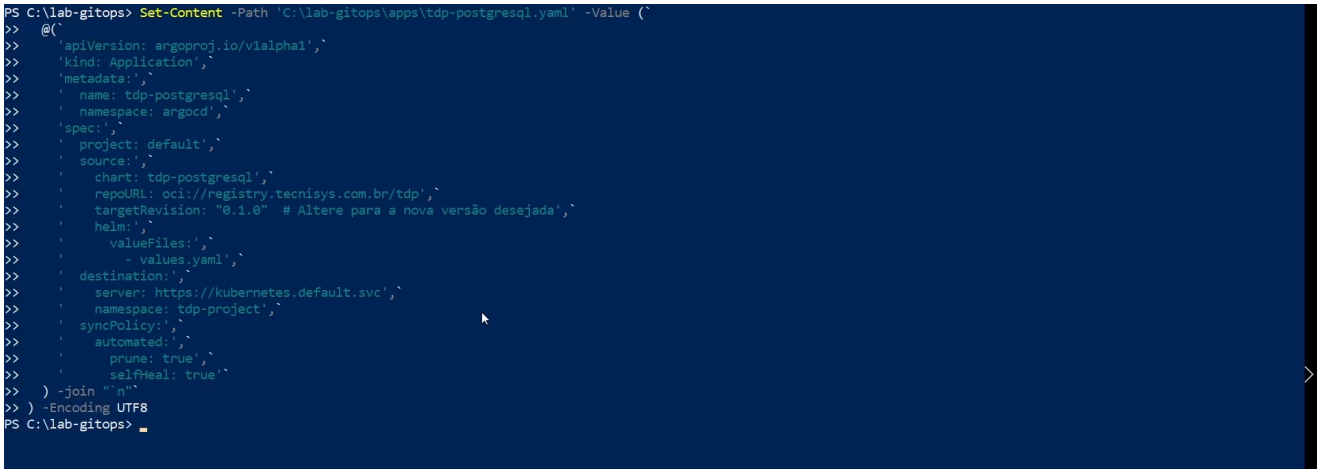


Figura 1 - Edição do Manifest

Alterar Valores de Configuração

Para atualizar valores de configuração junto com a versão, modifique o campo `helm.values` ou o arquivo `values.yaml` referenciado. Exemplo para o PostgreSQL (recursos, armazenamento, etc.):

```
spec:
  source:
    chart: tdp-postgresql
    repoURL: oci://registry.tecnisys.com.br/tdp
    targetRevision: 3.0.1
    helm:
      values: |
        primary:
          resources:
            requests:
              memory: 2Gi
              cpu: "1"
        persistence:
          size: 20Gi
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> cd C:\lab-gitops
>> $path = 'C:\lab-gitops\apps\tdp-postgresql.yaml'
>> $content = @(
>>   'apiVersion: argoproj.io/v1alpha1',
>>   'kind: Application',
>>   'metadata:',
>>   '  name: tdp-postgresql',
>>   '  namespace: argocd',
>>   'spec:',
>>   '  project: default',
>>   '  source:',
>>   '    chart: tdp-postgresql',
>>   '    repoURL: oci://registry.tecnisys.com.br/tdp',
>>   '    targetRevision: "0.1.0"',
>>   '    helm:',
>>   '      values: |',
>>   '        primary:',
>>   '          resources:',
>>   '            requests:',
>>   '              memory: 4Gi',
>>   '              cpu: "1"',
>>   '        persistence:',
>>   '          size: 25Gi',
>>   '  destination:',
>>   '    server: https://kubernetes.default.svc',
>>   '    namespace: tdp-project',
>>   '  syncPolicy:',
>>   '    automated:',
>>   '      prune: true',
>>   '      selfHeal: true'
>> ) -join "`r`n"
>> [System.IO.File]::WriteAllText($path, $content, [System.Text.UTF8Encoding]::new($false))
```

Figura 2 - Edição do Manifest - configuração

Confirmar as Alterações no Git

Após editar os manifests, faça commit e push das alterações para o repositório Git:

Terminal input

```
git add apps/
git commit -m "chore: atualizar tdp-postgresql para nova versão"
git push origin main
```

```
PS C:\lab-gitops> git add apps\tdp-postgresql.yaml
PS C:\lab-gitops> git commit -m "chore: aumentar recursos tdp-postgresql (memory 4Gi, persistence 25Gi)"
[main add4ab1] chore: aumentar recursos tdp-postgresql (memory 4Gi, persistence 25Gi)
 1 file changed, 12 insertions(+), 5 deletions(-)
PS C:\lab-gitops> git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 741 bytes | 741.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: . Processing 1 references
remote: Processed 1 references in total
To http://localhost:3001/admin/tdp-gitops-docs.git
   3e625e0..add4ab1  main -> main
PS C:\lab-gitops>
```

Figura 3 - Confirmar alteração no Git

O ArgoCD detectará automaticamente as alterações no repositório e iniciará o processo de sincronização, conforme a estratégia de sync configurada.

Estratégias de Sincronização

Sincronização Automática

© Copyright Tecnisys 2026. All rights reserved.

Com a sincronização automática habilitada, o ArgoCD aplica as alterações detectadas no Git sem intervenção manual:

```
syncPolicy:
  automated:
    prune: true      # Remove recursos que não estão mais no Git
    selfHeal: true   # Corrige desvios entre o estado real e o desejado
```

> **INFO**
>
> A sincronização automática é recomendada para ambientes de desenvolvimento e staging. Em produção, considere sincronização manual para maior controle.

Sincronização Manual

Para atualizações em produção, desabilite a sincronização automática e realize o sync manualmente:

```
syncPolicy: {}  # Sem política automática
```

Terminal input

```
argocd app sync tdp-postgresql
```

© Copyright Tecnisys 2026. All rights reserved.

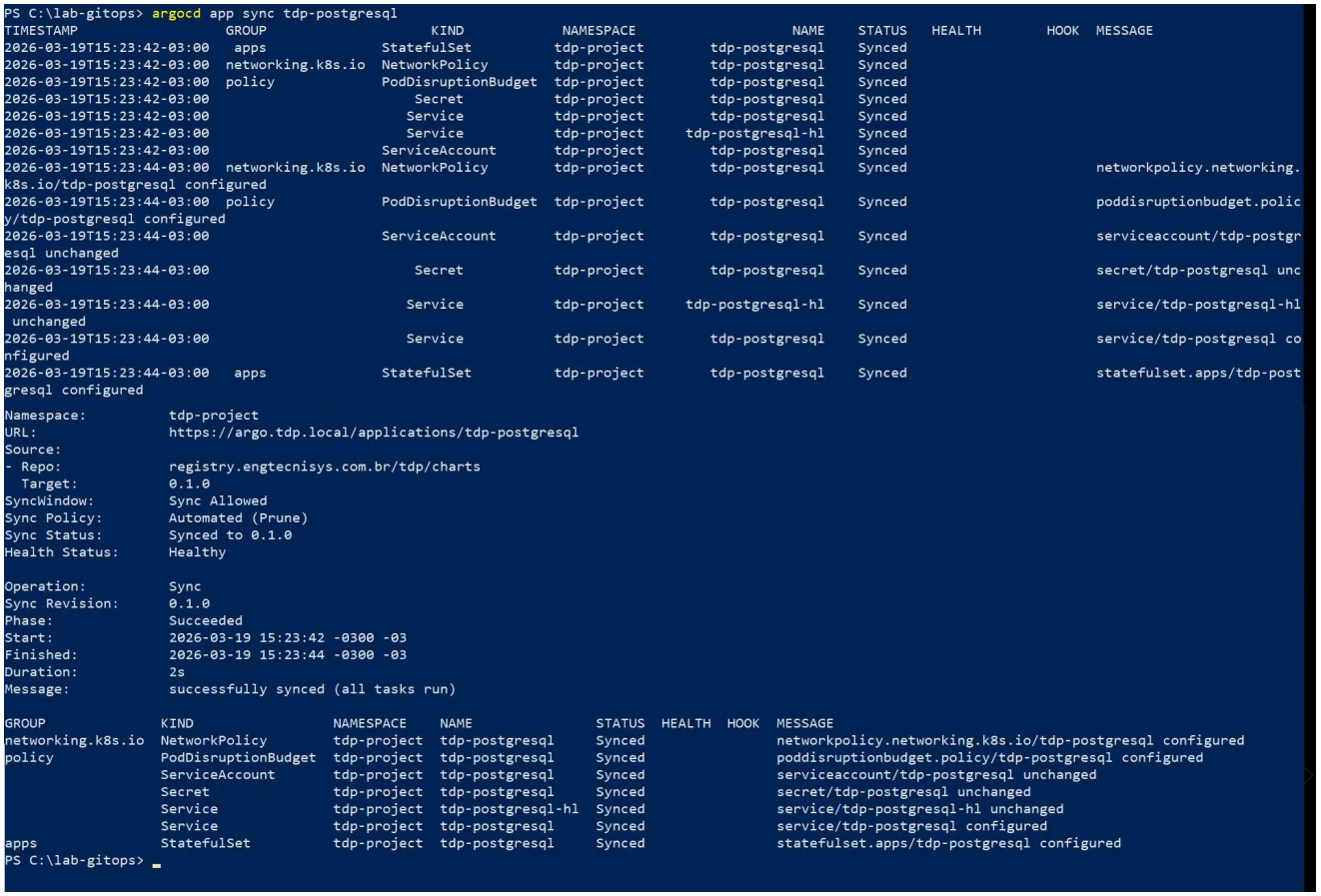


Figura 4 - Sincronização manual

Ou clique no botão **Sync** da Application na interface web do ArgoCD.

Sync Waves

Para controlar a ordem de atualização dos componentes, utilize **sync waves**. Recursos com valores menores de wave são sincronizados primeiro:

```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "1"
```

Wave	Componentes
0	CRDs (`tdp-crds`)

© Copyright Tecnisys 2026. All rights reserved.

Wave	Componentes
1	Infraestrutura (`tdp-argo`, `tdp-postgresql`)
2	Mensageria (`tdp-kafka`)
3	Armazenamento (`tdp-hive-metastore`, `tdp-iceberg`, `tdp-ozone`, `tdp-deltalake`)
4	Segurança (`tdp-ranger`)
5	Processamento (`tdp-nifi`, `tdp-spark`, `tdp-trino`, `tdp-clickhouse`)
6	Ferramentas (`tdp-airflow`, `tdp-jupyter`, `tdp-superset`, `tdp-openmetadata`, `tdp-cloudbeaver`)

Rollouts Progressivos

Dry Run

Simule a sincronização antes de aplicar para verificar o que será alterado:

Terminal input

```
argocd app sync tdp-postgresql --dry-run
```

Preview das Diferenças

Visualize as diferenças entre o estado atual do cluster e o estado desejado no Git:

Terminal input

```
argocd app diff tdp-postgresql
```

Sincronização Seletiva

© Copyright Tecnisys 2026. All rights reserved.

Sincronize apenas recursos específicos de uma Application, em vez de todos de uma vez:

Terminal input

```
argocd app sync tdp-postgresql --resource apps:StatefulSet:tdp-postgresql
```

Validação

Validação Pós-Atualização

Após aplicar as alterações no repositório Git, utilize os comandos abaixo para acompanhar o estado da sincronização.

Comandos

Verificar o status das Applications

Terminal input

```
argocd app list
```

```
PS C:\lab-gitops> argocd app list
NAME                  CLUSTER                         NAMESPACE    PROJECT  STATUS  HEALTH   SYNCPOLICY  CONDITIONS  REPO
                                                PATH  TARGET
argocd/tdp-postgresql  https://kubernetes.default.svc  tdp-project  default  Synced  Healthy  Auto-Prune  <none>      registry.tecnisys.com.br/tdp
/charts                                         3.0.0
argocd/tdp-stack       https://kubernetes.default.svc  argocd       default  Synced  Healthy  Auto-Prune  <none>      http://gitea-http.gitea.svc.
cluster.local:3000/admin/tdp-gitops-docs.git  apps  main
PS C:\lab-gitops>
```

Figura 1 - Applications do TDP e seus status

Status de Sincronização	Descrição
Synced	O estado do cluster corresponde ao declarado no manifest
OutOfSync	Existem diferenças entre o manifest e o cluster

© Copyright Tecnisys 2026. All rights reserved.

Status de Sincronização	Descrição
Unknown	O ArgoCD não conseguiu determinar o status
Status de Saúde	Descrição
Healthy	Todos os recursos da Application estão operacionais
Progressing	Os recursos estão sendo criados ou atualizados
Degraded	Um ou mais recursos apresentam problemas
Suspended	A sincronização está pausada

Verificar uma Application específica

Terminal input

```
argocd app get tdp-postgresql
```

© Copyright Tecnisys 2026. All rights reserved.

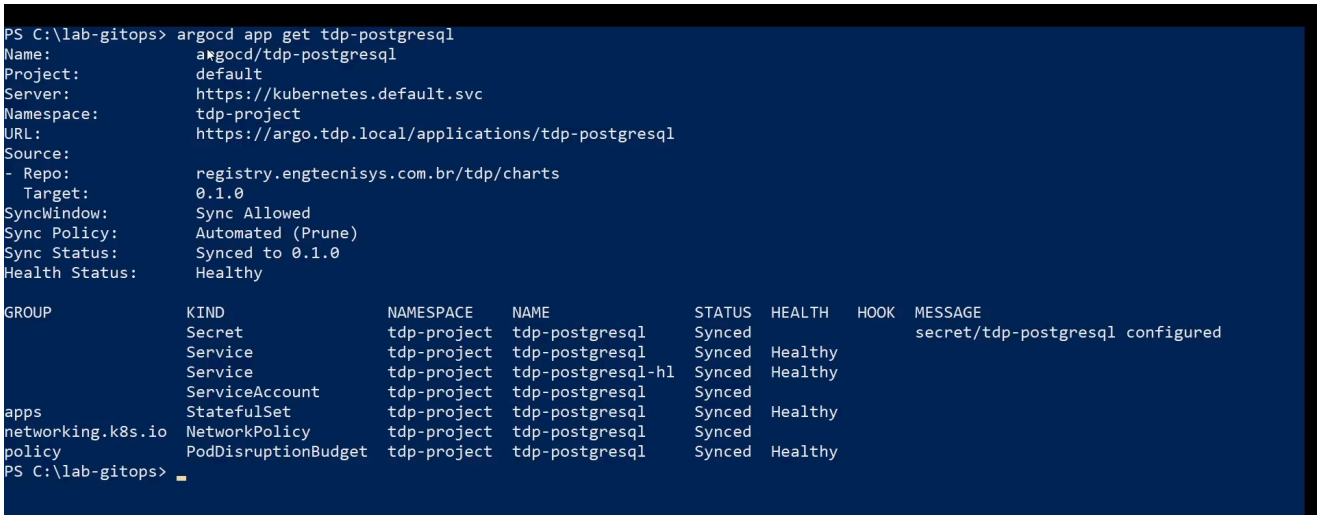


Figura 2 - Verificar a application postgresql

Para acompanhar o progresso em tempo real:

Terminal input

```
argocd app wait tdp-postgresql --sync
```

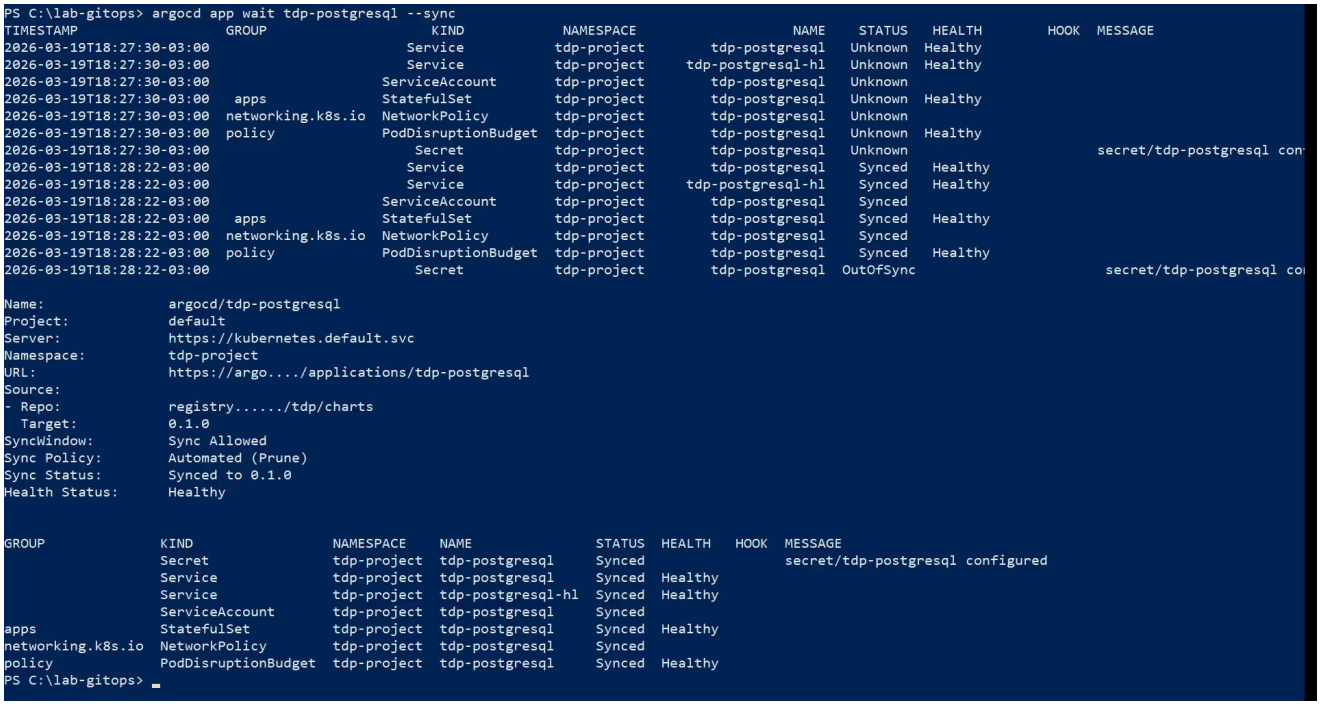


Figura 3 - Acompanhar o progresso em tempo real

Verificar a saúde dos recursos

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
argocd app wait tdp-postgresql --health
```

```
PS C:\lab-gitops\tdp-gitops-docs> argocd app wait tdp-postgresql --health

Name:               argocd/tdp-postgresql
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          tdp-project
URL:                http://argo.tdp.local/applications/tdp-postgresql
Source:
- Repo:             registry.engtecnisys.com.br/tdp/charts
  Target:           0.1.0
SyncWindow:         Sync Allowed
Sync Policy:        Automated (Prune)
Sync Status:        Synced to 0.1.0
Health Status:      Healthy

GROUP               KIND                  NAMESPACE    NAME               STATUS  HEALTH   HOOK  MESSAGE
networking.k8s.io   NetworkPolicy         tdp-project  tdp-postgresql     Synced                 networkpolicy.networking.k8s.io/tdp-post
gresql configured
policy              PodDisruptionBudget   tdp-project  tdp-postgresql     Synced  Healthy        poddisruptionbudget.policy/tdp-postgresq
l configured
                    ServiceAccount        tdp-project  tdp-postgresql     Synced                 serviceaccount/tdp-postgresql unchanged
                    Secret                tdp-project  tdp-postgresql     Synced                 secret/tdp-postgresql unchanged
                    Service               tdp-project  tdp-postgresql-hl  Synced  Healthy        service/tdp-postgresql-hl unchanged
                    Service               tdp-project  tdp-postgresql     Synced  Healthy        service/tdp-postgresql configured
apps                StatefulSet           tdp-project  tdp-postgresql     Synced  Healthy        statefulset.apps/tdp-postgresql configur
ed
PS C:\lab-gitops\tdp-gitops-docs>
```

Figura 4 - Verificar a saude dos recursos

Verificar os pods e a prontidão dos workloads

Terminal input

```
kubectl get pods -n <NAMESPACE>
```

```
PS C:\lab-gitops> kubectl get pods -n tdp-project
NAME               READY   STATUS    RESTARTS      AGE
tdp-postgresql-0   1/1     Running   2 (62m ago)   3h5m
PS C:\lab-gitops>
```

Figura 5 - Verificar os pods

Terminal input

```
kubectl get statefulset -n <NAMESPACE>
```

```
PS C:\lab-gitops> kubectl get statefulset -n tdp-project
NAME                      READY   AGE
demo-list-tdp-postgresql  1/1     4d13h
tdp-postgresql            1/1     4d16h
PS C:\lab-gitops>
```

© Copyright Tecnisys 2026. All rights reserved.

Figura 6 - Verificar os StatefulSets

Se a aplicação utilizar Deployment, valide também:

Terminal input

```
kubectl get deploy -n <NAMESPACE>
```

O que confirmar:

- Os pods estão com `STATUS` **Running**
- A coluna `READY` dos pods está completa, por exemplo `1/1`
- Os StatefulSets ou Deployments da aplicação mostram todas as réplicas prontas, por exemplo `1/1`, `2/2` ou `3/3`

Verificar a Revisão do Helm

Confirme que a revisão do release foi incrementada com sucesso:

Terminal input

```
helm history <RELEASE_NAME> -n <NAMESPACE>
```

```
PS C:\lab-gitops\tdp-gitops-docs> helm history demo-list -n tdp-project
REVISION        UPDATED                         STATUS          CHART                   APP VERSION     DESCRIPTION
1               Wed Mar 18 20:18:01 2026        deployed        tdp-postgresql-0.1.0    17.5.0          Install complete
PS C:\lab-gitops\tdp-gitops-docs>
```

Figura 7 - Verificar a Revisão do Helm

Resolução de Problemas

Resolução de Problemas

Comandos

Application em OutOfSync ou Degraded

Se uma Application apresenta status `OutOfSync` ou `Degraded`:

© Copyright Tecnisys 2026. All rights reserved.

1. Verifique os detalhes e erros da operação:

Terminal input

```
argocd app get tdp-postgresql --show-operation
```

2. Analise os logs do controller do ArgoCD:

Terminal input

```
kubectl logs -n argocd -l app.kubernetes.io/name=argocd-application-
controller --tail=200
```

3. Tente sincronizar manualmente:

Terminal input

```
argocd app sync tdp-postgresql
```

Conflitos de Sincronização

Se houver recursos modificados manualmente no cluster que conflitam com o Git:

Opção 1 — Forçar sincronização (sobrescreve o cluster com o Git):

Terminal input

```
argocd app sync tdp-postgresql --force
```

Opção 2 — Self-heal (deixa o ArgoCD corrigir automaticamente os desvios):

```
syncPolicy:
  automated:
    selfHeal: true
```

© Copyright Tecnisys 2026. All rights reserved.

Retry Automático

Configure o retry automático para que o ArgoCD tente novamente em caso de falhas temporárias:

```
syncPolicy:
  retry:
    limit: 5
    backoff:
      duration: 5s
      factor: 2
      maxDuration: 3m
```

Reverter a Atualização

Se a atualização falhar e for necessário reverter, altere o `targetRevision` no manifest Git de volta para a versão anterior e faça commit:

Terminal input

```
git revert HEAD
git push origin main
```

O ArgoCD detectará a reversão e sincronizará o cluster para o estado anterior.

> **TIP**
>
> Para mais detalhes sobre rollback, consulte a página de Rollback.

© Copyright Tecnisys 2026. All rights reserved.

Rollback

Este guia descreve os procedimentos para reverter atualizações dos componentes do TDP Kubernetes, tanto via Helm quanto via ArgoCD. O rollback permite restaurar rapidamente uma versão anterior em caso de falhas ou comportamentos inesperados após uma atualização.

> **ATENÇÃO**
>
> Antes de realizar qualquer rollback, faça backup dos dados persistentes. Dependendo do componente, um rollback pode causar perda de dados se as migrações de schema forem incompatíveis com versões anteriores.

Backup de Dados antes do Rollback

Antes de iniciar o rollback, proteja os dados persistentes criando snapshots dos PVCs.

Identificar os PVCs do Componente

Terminal input

```
kubectl get pvc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

Criar Snapshots dos Volumes

Terminal input

```
kubectl apply -f - <<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: <RELEASE_NAME>-pre-rollback-snapshot
  namespace: <NAMESPACE>
spec:
  volumeSnapshotClassName: <snapshot-class>
  source:
```

© Copyright Tecnisys 2026. All rights reserved.

```
    persistentVolumeClaimName: <pvc-name>
EOF
```

Verificar o Status do Snapshot

Terminal input

```
kubectl get volumesnapshot -n <NAMESPACE>
```

Certifique-se de que o snapshot está com status `readyToUse: true` antes de prosseguir.

Rollback via Helm

Consultar o Histórico de Revisões

Utilize o comando `helm history` para visualizar todas as revisões de um release:

Terminal input

```
helm history <RELEASE_NAME> -n <NAMESPACE>
```

A saída exibirá uma tabela com as colunas: `REVISION`, `UPDATED`, `STATUS`, `CHART`, `APP VERSION` e `DESCRIPTION`. Identifique a revisão para a qual deseja reverter.

Exemplo de saída:

```
REVISION    UPDATED                     STATUS        CHART              APP
VERSION    DESCRIPTION
1           2025-01-15 10:30:00         deployed      tdp-kafka-1.0.0    3.5.1
Install complete
2           2025-02-10 14:00:00         superseded    tdp-kafka-1.0.0    3.5.1
Upgrade complete
3           2025-03-01 09:15:00         deployed      tdp-kafka-2.0.0    3.6.0
Upgrade complete
```

Rollback para uma Revisão Específica

Para reverter para uma revisão específica:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
helm rollback <RELEASE_NAME> <revisao> -n <NAMESPACE>
```

Por exemplo, para reverter o `tdp-kafka` para a revisão 2:

Terminal input

```
helm rollback tdp-kafka 2 -n <NAMESPACE>
```

Rollback para a Revisão Anterior

Para reverter para a revisão imediatamente anterior (sem especificar o número):

Terminal input

```
helm rollback <RELEASE_NAME> -n <NAMESPACE>
```

Rollback com Recriação de Pods

Em alguns casos, pode ser necessário forçar a recriação de todos os pods durante o rollback:

Terminal input

```
helm rollback <RELEASE_NAME> <revisao> -n <NAMESPACE> --recreate-pods
```

Verificar o Resultado do Rollback

Após o rollback, confirme que o release foi revertido com sucesso:

Terminal input

```
helm status <RELEASE_NAME> -n <NAMESPACE>
helm history <RELEASE_NAME> -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

A última entrada no histórico deve indicar `STATUS: deployed` com a descrição `Rollback to <revisao>`.

Rollback via ArgoCD

O rollback via ArgoCD consiste em reverter o estado do repositório Git para a versão anterior dos manifestos.

Reverter o Manifesto no Git

A abordagem recomendada é reverter o commit que introduziu a atualização:

Terminal input

```
git log --oneline -10  # Identifique o commit da atualização
git revert <commit-hash>
git push origin main
```

O ArgoCD detectará a alteração no repositório e sincronizará o cluster automaticamente (se a sincronização automática estiver habilitada).

Forçar Sincronização após Reversão

Se a sincronização automática estiver desabilitada, force a sincronização manualmente:

Terminal input

```
argocd app sync tdp-kafka
```

Rollback pelo ArgoCD CLI

O ArgoCD também permite reverter a última operação de sincronização:

Terminal input

```
argocd app rollback tdp-kafka
```

NOTA

© Copyright Tecnisys 2026. All rights reserved.

> O rollback pelo ArgoCD CLI reverte apenas o estado do cluster. O repositório Git não é alterado, o que pode causar um novo sync automático restaurando a versão indesejada. Sempre reverta também no Git para garantir consistência.

Verificar o Estado após Rollback

Terminal input

```
argocd app get tdp-kafka
```

O status deve indicar `Synced` e `Healthy` após a conclusão do rollback.

Cenários Comuns de Rollback

Cenário 1: Falha na Inicialização dos Pods

Sintoma: pods ficam em estado `CrashLoopBackOff` ou `Error` após atualização.

Diagnóstico:

Terminal input

```
kubectl describe pod -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME> --
previous
```

Solução: rollback para a versão anterior:

Terminal input

```
helm rollback <RELEASE_NAME> -n <NAMESPACE>
```

Cenário 2: Incompatibilidade de Schema do Banco de Dados

Sintoma: erros de migração de schema nos logs do componente.

© Copyright Tecnisys 2026. All rights reserved.

Diagnóstico:

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME> | grep -i "migration\|schema\|database"
```

Solução:

1. Restaure o backup do banco de dados (snapshot do PVC)
2. Realize o rollback do release:

   Terminal input

   ```
   helm rollback <RELEASE_NAME> -n <NAMESPACE>
   ```

3. Verifique se o componente inicializa corretamente com o banco restaurado

Cenário 3: Problemas de Conectividade entre Componentes

Sintoma: componentes não conseguem se comunicar após atualização parcial.

Diagnóstico:

Terminal input

```
kubectl get endpoints -n <NAMESPACE>
kubectl get svc -n <NAMESPACE>
```

Solução: verifique se todos os componentes dependentes foram atualizados para versões compatíveis. Se necessário, realize rollback de todos os componentes afetados.

Cenário 4: Degradação de Performance

Sintoma: latência elevada ou consumo excessivo de recursos após atualização.

Diagnóstico:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl top pods -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
kubectl describe pod -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

Solução: verifique as diferenças de configuração entre as versões e ajuste os `resources` no `values.yaml`. Se o problema persistir, realize rollback.

Rollback Parcial (Componente Individual)

Em cenários onde apenas um componente apresenta problemas após uma atualização geral, é possível realizar rollback apenas desse componente.

Identificar o Componente Problemático

Terminal input

```
kubectl get pods -n <NAMESPACE> --field-selector=status.phase!=Running
helm list -n <NAMESPACE> --failed
```

Rollback Individual via Helm

Terminal input

```
helm rollback <release-problematico> -n <NAMESPACE>
```

Verificar Compatibilidade

Após o rollback parcial, verifique se o componente revertido é compatível com as demais versões atualizadas:

Terminal input

```
kubectl logs -n <NAMESPACE> -l app.kubernetes.io/instance=<release-problematico>
--tail=50
kubectl get pods -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.



> **ATENÇÃO**
>
> O rollback parcial pode causar incompatibilidades entre componentes se houver dependências de versão. Consulte a matriz de compatibilidade antes de prosseguir.

Boas Práticas de Rollback

1. **Sempre faça backup** dos dados persistentes antes de iniciar o rollback
2. **Documente a causa** do rollback para referência futura
3. **Teste em ambiente de staging** antes de atualizar produção, reduzindo a necessidade de rollbacks
4. **Monitore os logs** após o rollback para confirmar a estabilidade
5. **Mantenha o Git consistente** com o estado do cluster -- se usar ArgoCD, sempre reverta no Git
6. **Não pule revisões** -- em caso de múltiplos rollbacks, reverta uma revisão de cada vez

© Copyright Tecnisys 2026. All rights reserved.

DESINSTALAÇÃO

© Copyright Tecnisys 2026. All rights reserved.

Desinstalação

Este guia descreve como remover componentes do TDP Kubernetes de forma organizada e segura.

Antes de começar

Distinguir o que está sendo removido:

- **componentes instalados diretamente via Helm**
- **Applications gerenciadas pelo ArgoCD**
- **o próprio ArgoCD**, quando ele também fizer parte da remoção

Este documento separa claramente cada cenário.

> **ATENÇÃO**
>
> A remoção dos recursos Kubernetes não implica, automaticamente, a remoção dos dados persistidos.
> PVCs, PVs, snapshots e namespace devem ser avaliados separadamente.

Antes de executar qualquer remoção, identifique **como o componente ou ambiente foi implantado**.

1. Verificar se o componente foi instalado via Helm

Use este comando para descobrir releases Helm existentes no namespace:

Terminal input

```
helm list -n <NAMESPACE>
```

```
PS C:\lab-gitops> helm list -n tdp-uninstall-doc
NAME            NAMESPACE              REVISION        UPDATED                                 STATUS          CHART
demo-uninstall  tdp-uninstall-doc       1               2026-03-25 07:55:41.058865 -0300 -03    deployed        tdp-postgresql-
PS C:\lab-gitops>
```

Figura 1 - Releases Helm existentes

© Copyright Tecnisys 2026. All rights reserved.

Se o componente aparecer nessa listagem, a remoção principal é feita com `helm uninstall`.

2. Verificar se o componente é gerenciado pelo ArgoCD

Use este comando para descobrir Applications gerenciadas pelo ArgoCD:

Terminal input

```
argocd app list
```

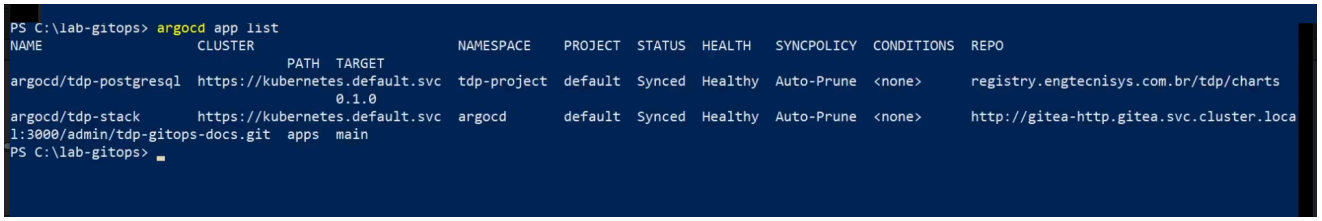


Figura 2 - Applications gerenciadas pelo ArgoCD

Se o componente aparecer como Application do ArgoCD, a remoção principal é feita com `argocd app delete`.

> **REGRA PRÁTICA**
>
> Não misture os fluxos sem necessidade.
> Se o componente é gerenciado pelo ArgoCD, o caminho principal é remover a Application.
> Se ele foi instalado diretamente via Helm, o caminho principal é remover a release Helm.

Preparações antes da remoção

Fazer backup dos dados

Antes de remover qualquer componente com estado, faça backup do que precisar ser preservado.

Exemplos comuns:

- PostgreSQL: dump do banco

© Copyright Tecnisys 2026. All rights reserved.

- ClickHouse: exportação de tabelas e schemas
- Airflow: DAGs, variáveis e conexões
- OpenMetadata: metadados e lineage
- Ranger: políticas e configurações relevantes

PostgreSQL — backup completo

Terminal input

```
kubectl exec -n <NAMESPACE> <POD_POSTGRESQL> -- \
  pg_dumpall -U postgres > backup-postgres.sql
```

Substitua `<POD_POSTGRESQL>` pelo nome do pod (obtenha com `kubectl get pods -n <NAMESPACE> -l app.kubernetes.io/name=postgresql`).

Exportar a configuração da release Helm

Quando o componente tiver sido instalado via Helm, exporte os valores atualmente aplicados:

Terminal input

```
helm get values <RELEASE_NAME> -n <NAMESPACE> -o yaml > <RELEASE_NAME>-values-backup.yaml
```

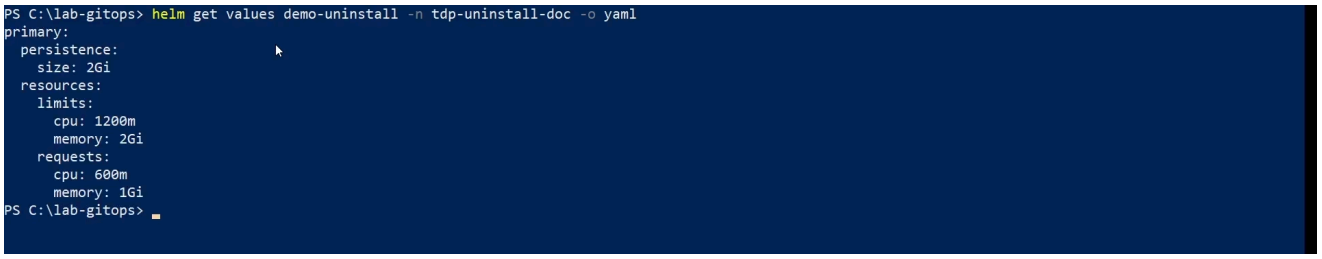


Figura 3 - Exportação da configuração da release Helm

> **NOTA**
>
> Esse arquivo normalmente contém os valores sobrescritos durante a instalação ou atualização.

© Copyright Tecnisys 2026. All rights reserved.

> Para consultar todos os parâmetros aceitos pelo chart, use `helm show values` no chart correspondente.

Para exportar os valores de todas as releases do namespace de uma só vez:

Terminal input

```
for release in $(helm list -n <NAMESPACE> -q); do
  helm get values "$release" -n <NAMESPACE> -o yaml > "${release}-values-backup.yaml"
  echo "Backup de $release concluído"
done
```

Identificar PVCs associados

Antes de remover o componente, verifique se existem volumes persistentes associados a ele:

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

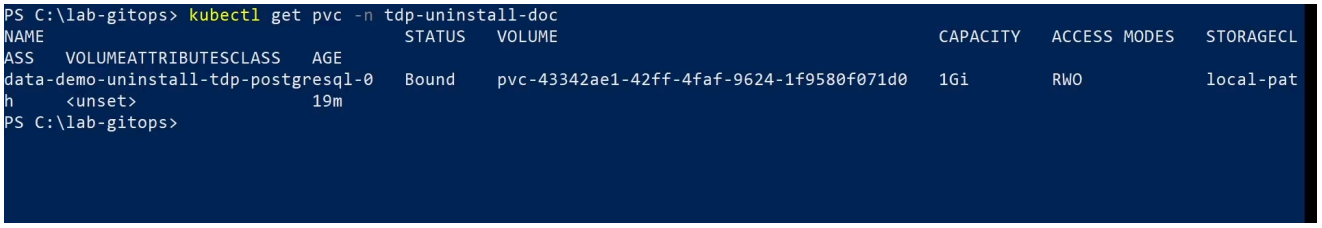


Figura 4 - PVCs associados à release

Para restringir a consulta a uma release Helm específica:

Terminal input

```
kubectl get pvc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

Criar snapshot de volumes persistentes, quando suportado

© Copyright Tecnisys 2026. All rights reserved.

Se o cluster suportar CSI snapshots e houver `VolumeSnapshotClass` disponível, crie snapshots dos PVCs críticos antes da remoção.

Verifique se o cluster suporta snapshots:

Terminal input

```
kubectl get volumesnapshotclass
```

Crie o snapshot:

Terminal input

```
kubectl apply -f - <<EOT
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: <pvc-name>-final-snapshot
  namespace: <NAMESPACE>
spec:
  volumeSnapshotClassName: <snapshot-class>
  source:
    persistentVolumeClaimName: <pvc-name>
EOT
```

Verifique o resultado:

Terminal input

```
kubectl get volumesnapshot -n <NAMESPACE>
```

Avaliar CRDs antes de remover a plataforma

O chart `tdp-crds` instala recursos de escopo global no cluster. Esses recursos não pertencem a um namespace específico e podem continuar existindo após a remoção das releases Helm ou Applications do Argo CD.

© Copyright Tecnisys 2026. All rights reserved.

Antes de remover CRDs, confirme que todos os componentes que dependem deles já foram removidos e que não há recursos customizados restantes:

Terminal input

```
kubectl get crd
```

Para avaliar objetos ainda existentes, consulte os tipos instalados pelos operadores usados na plataforma, como Argo CD, Strimzi, NiFiKop, Altinity ClickHouse Operator, OpenMetadata OMJob Operator e cert-manager.

> **CAUTION**
>
> Remova CRDs apenas em cenários de desinstalação completa da plataforma ou limpeza administrativa controlada. Remover um CRD também remove a API customizada correspondente e pode afetar recursos ainda existentes no cluster.

Remover componentes

Via Helm

Comandos

Desinstalar um componente individual

Use este fluxo quando o componente tiver sido instalado diretamente como release Helm.

Desinstalar a release

Terminal input

```
helm uninstall <RELEASE_NAME> -n <NAMESPACE>
```



© Copyright Tecnisys 2026. All rights reserved.

Figura 5 - Desinstalação de um componente via Helm

Verificar se os recursos da release foram removidos

Terminal input

```
kubectl get all -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

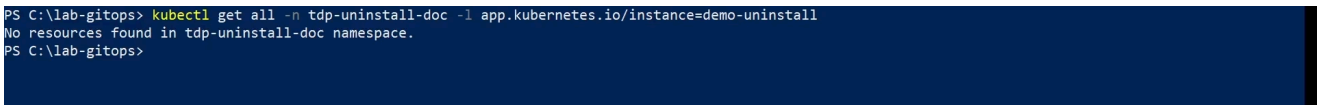


Figura 6 - Recursos gerenciados pela release após a desinstalação

> **NOTE**
>
> Esse comando mostra o que ainda está associado à release após o `helm uninstall`.
> Se o resultado vier vazio, os recursos foram removidos com sucesso.

Verificar PVCs remanescentes

Terminal input

```
kubectl get pvc -n <NAMESPACE> -l app.kubernetes.io/instance=<RELEASE_NAME>
```

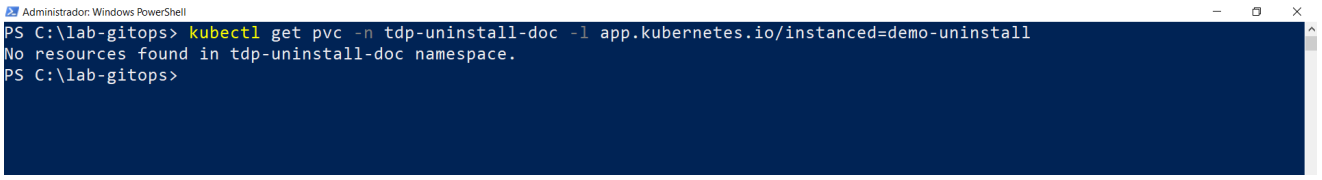


Figura 7 - PVCs remanescentes

> **NOTA**
>
> O `helm uninstall` remove a release e seus recursos Kubernetes associados — Deployments, StatefulSets, Services, ConfigMaps e Secrets.
> Os PVCs normalmente **não** são removidos automaticamente.

Remover o ambiente completo via Helm

© Copyright Tecnisys 2026. All rights reserved.



Use este fluxo quando o ambiente inteiro tiver sido implantado diretamente via Helm, sem Applications gerenciadas pelo ArgoCD.

Remover as releases em ordem reversa de dependência

Terminal input

```
helm uninstall tdp-superset -n <NAMESPACE>
helm uninstall tdp-openmetadata -n <NAMESPACE>
helm uninstall tdp-cloudbeaver -n <NAMESPACE>
helm uninstall tdp-jupyter -n <NAMESPACE>
helm uninstall tdp-airflow -n <NAMESPACE>
helm uninstall tdp-clickhouse -n <NAMESPACE>
helm uninstall tdp-trino -n <NAMESPACE>
helm uninstall tdp-spark -n <NAMESPACE>
helm uninstall tdp-nifi -n <NAMESPACE>
helm uninstall tdp-ranger -n <NAMESPACE>
helm uninstall tdp-iceberg -n <NAMESPACE>
helm uninstall tdp-deltalake -n <NAMESPACE>
helm uninstall tdp-ozone -n <NAMESPACE>
helm uninstall tdp-hive-metastore -n <NAMESPACE>
```

KAFKA E CRDS DO STRIMZI

Antes de desinstalar o `tdp-kafka`, verifique se há recursos Strimzi pendentes no namespace:

Terminal input

```
kubectl get kafkas,kafkatopics,kafkausers,kafkaconnects,kafkaconnectors
-n <NAMESPACE>
```

Remova os recursos customizados antes de desinstalar o chart para evitar CRDs órfãos no cluster.

Terminal input

```
helm uninstall tdp-kafka -n <NAMESPACE>
helm uninstall tdp-postgresql -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
helm uninstall tdp-argo -n <NAMESPACE>
helm uninstall tdp-crds -n <NAMESPACE>
```

Verificar PVCs restantes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

Remover PVCs, se a remoção for definitiva

Terminal input

```
kubectl delete pvc --all -n <NAMESPACE>
```

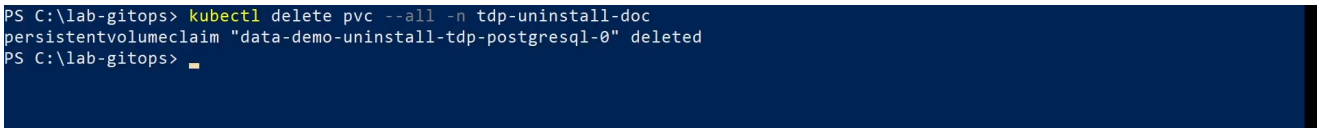


Figura 8 - Remoção definitiva PVCs

> **PERDA DE DADOS**
>
> Excluir PVCs remove permanentemente os dados persistidos.
> Faça isso apenas quando a desmontagem for definitiva e os backups já estiverem concluídos.

Via ArgoCD

Comandos

Remover um componente individual via ArgoCD

Use este fluxo quando o componente for gerenciado como Application do ArgoCD.

Remover a Application

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
argocd app delete <componente> --cascade
```



Figura 9 - Remoção de uma Application no ArgoCD

Verificar a remoção

Terminal input

```
argocd app list
```

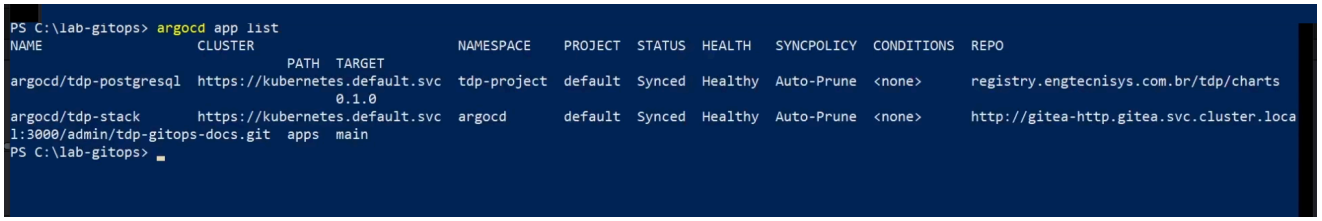


Figura 10 - Verificação da remoção no ArgoCD

Verificar recursos restantes no namespace

Terminal input

```
kubectl get all -n <NAMESPACE>
```



Figura 11 - Verificação de recursos restantes no Namespace

Verificar PVCs restantes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> kubectl get pvc -n tdp-uninstall-doc
NAME                                     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECL
ASS   VOLUMEATTRIBUTESCLASS   AGE
data-demo-uninstall-tdp-postgresql-0   Bound    pvc-43342ae1-42ff-4faf-9624-1f9580f071d0   1Gi        RWO            local-pat
h     <unset>                 19m
PS C:\lab-gitops>
```

Figura 12 - PVCs restantes

> **ATENÇÃO**
>
> A opção `--cascade` remove a Application e os recursos Kubernetes gerenciados por ela.
> Mesmo assim, PVCs, PVs e outros resíduos de infraestrutura podem exigir verificação e limpeza adicional.

Remover o ambiente completo via ArgoCD

Use este caminho quando o ambiente TDP estiver inteiramente implantado e gerenciado pelo ArgoCD.

Opção A — Remover uma app raiz do tipo app-of-apps

Se o ambiente utilizar uma Application raiz que controla as demais, a remoção pode começar por ela:

Terminal input

```
argocd app delete <root-application> --cascade
```

Use esta opção apenas quando o modelo app-of-apps estiver validado no seu ambiente e você tiver certeza de que a exclusão da app raiz removerá corretamente as aplicações filhas e seus recursos.

Opção B — Remover as Applications individualmente em ordem reversa de dependência

Quando não houver app raiz, ou quando você quiser controle total da desmontagem, remova as Applications uma a uma:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
argocd app delete tdp-superset --cascade
argocd app delete tdp-openmetadata --cascade
argocd app delete tdp-cloudbeaver --cascade
argocd app delete tdp-jupyter --cascade
argocd app delete tdp-airflow --cascade
argocd app delete tdp-clickhouse --cascade
argocd app delete tdp-trino --cascade
argocd app delete tdp-spark --cascade
argocd app delete tdp-nifi --cascade
argocd app delete tdp-ranger --cascade
argocd app delete tdp-iceberg --cascade
argocd app delete tdp-deltalake --cascade
argocd app delete tdp-ozone --cascade
argocd app delete tdp-hive-metastore --cascade
argocd app delete tdp-kafka --cascade
argocd app delete tdp-postgresql --cascade
```

Verificar se ainda restam Applications

Terminal input

```
argocd app list
```

Verificar se ainda restam recursos no namespace

Terminal input

```
kubectl get all -n <NAMESPACE>
```

Verificar PVCs restantes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

Remover PVCs, se a remoção for definitiva

Terminal input

```
```

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl delete pvc --all -n <NAMESPACE>
```

> **PERDA DE DADOS**
>
> Excluir PVCs remove permanentemente os dados persistidos.
> Faça isso apenas quando a desmontagem for definitiva e os backups já estiverem concluídos.

Remover a infraestrutura do ArgoCD

Esta etapa é separada da remoção do ambiente TDP.

Primeiro remova todas as Applications e os workloads gerenciados pelo ArgoCD.
Só depois remova o próprio ArgoCD, se isso fizer parte do objetivo.

O ArgoCD é removido via Helm, independentemente de como o ambiente TDP foi gerenciado.

Desinstalar as releases do ArgoCD

Terminal input

```
helm uninstall tdp-argo -n <NAMESPACE>
helm uninstall tdp-crds -n <NAMESPACE>
```

> **ATENÇÃO**
>
> O `helm uninstall tdp-crds` **não remove os CRDs automaticamente**. Os CRDs são recursos de escopo de cluster e podem ser compartilhados por outras instalações ou projetos. O `tdp-crds` gerencia CRDs de múltiplos componentes (Argo CD, Kafka, NiFiKop, ClickHouse, OpenMetadata e cert-manager) — avalie com cuidado antes de removê-los.

Remover CRDs manualmente (quando necessário)

Aplique o Job de cleanup incluído no chart:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
kubectl apply -f stack/charts/tdp/tdp-crds/cleanup/delete-crds-job.yaml
```

O Job verifica se existem recursos dependentes antes de deletar cada CRD — CRDs com recursos em uso são ignorados.

Limpeza complementar

Verificar Persistent Volumes órfãos

Após remover os PVCs, verifique se existem PVs em estado `Released`:

Terminal input

```
kubectl get pv | grep <NAMESPACE>
```

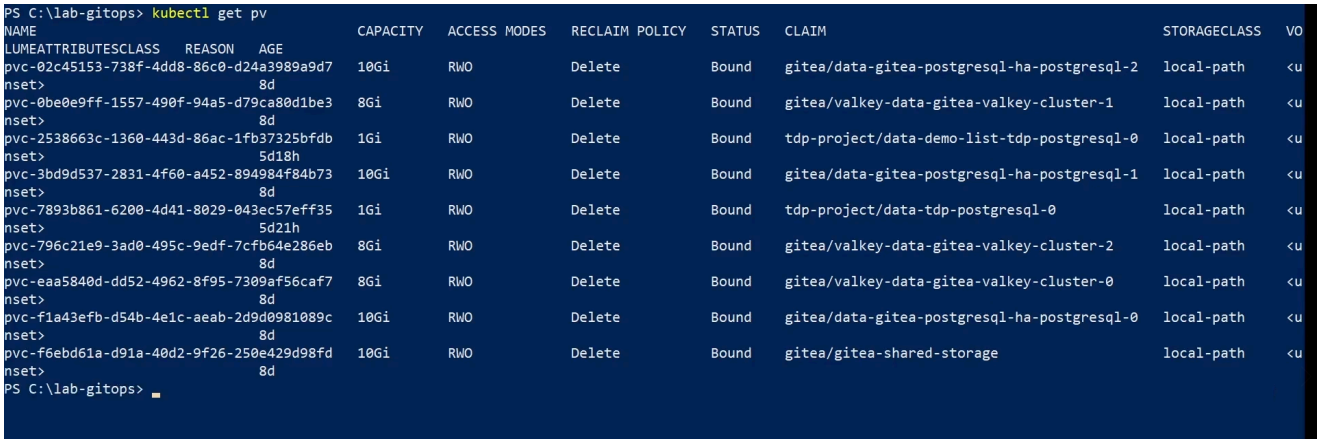


Figura 13 - Verificação de Persistent Volumes órfãos

Se houver PVs órfãos e não precisar mais deles:

Terminal input

```
kubectl delete pv <pv-name>
```

© Copyright Tecnisys 2026. All rights reserved.

```
PS C:\lab-gitops> kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                                  STORAGECLASS   VO
LUMEATTRIBUTESCLASS   REASON   AGE
pvc-02c45153-738f-4dd8-86c0-d24a3989a9d7   10Gi       RWO            Delete           Bound    gitea/data-gitea-postgresql-ha-postgresql-2            local-path     <u
nset>                        8d
pvc-0be0e9ff-1557-490f-94a5-d79ca80d1be3   8Gi        RWO            Delete           Bound    gitea/valkey-data-gitea-valkey-cluster-1               local-path     <u
nset>                        8d
pvc-2538663c-1360-443d-86ac-1fb37325bfdb   1Gi        RWO            Delete           Bound    tdp-project/data-demo-list-tdp-postgresql-0            local-path     <u
nset>                        5d18h
pvc-3bd9d537-2831-4f60-a452-894984f84b73   10Gi       RWO            Delete           Bound    gitea/data-gitea-postgresql-ha-postgresql-1            local-path     <u
nset>                        8d
pvc-7893b861-6200-4d41-8029-043ec57eff35   1Gi        RWO            Delete           Bound    tdp-project/data-tdp-postgresql-0                      local-path     <u
nset>                        5d21h
pvc-796c21e9-3ad0-495c-9edf-7cfb64e286eb   8Gi        RWO            Delete           Bound    gitea/valkey-data-gitea-valkey-cluster-2               local-path     <u
nset>                        8d
pvc-eaa5840d-dd52-4962-8f95-7309af56caf7   8Gi        RWO            Delete           Bound    gitea/valkey-data-gitea-valkey-cluster-0               local-path     <u
nset>                        8d
pvc-f1a43efb-d54b-4e1c-aeab-2d9d0981089c   10Gi       RWO            Delete           Bound    gitea/data-gitea-postgresql-ha-postgresql-0            local-path     <u
nset>                        8d
pvc-f6ebd61a-d91a-40d2-9f26-250e429d98fd   10Gi       RWO            Delete           Bound    gitea/gitea-shared-storage                             local-path     <u
nset>                        8d
PS C:\lab-gitops>
```

Figura 14 - Delete PVs

Remover o namespace, quando aplicável

Remova o namespace apenas quando ele for dedicado exclusivamente ao ambiente que está sendo desmontado.

Terminal input

```
kubectl delete namespace <NAMESPACE>
```

> **ATENÇÃO**
>
> A remoção do namespace exclui todos os recursos namespaced ainda existentes nele, incluindo PVCs, ConfigMaps, Secrets e ServiceAccounts.

Se o namespace ficar preso em `Terminating`, verifique finalizers:

Terminal input

```
kubectl get namespace <NAMESPACE> -o json | jq '.spec.finalizers'
```

Para forçar a finalização:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
kubectl get namespace <NAMESPACE> -o json | jq '.spec.finalizers = []' | \
kubectl replace --raw "/api/v1/namespaces/<NAMESPACE>/finalize" -f -
```

Verificação final

Após a remoção, valide se ainda restou algo no ambiente.

Verificar recursos restantes no namespace

Terminal input

```
kubectl get all -n <NAMESPACE>
```

Verificar releases Helm restantes

Terminal input

```
helm list -n <NAMESPACE>
```

Verificar Applications restantes no ArgoCD

Terminal input

```
argocd app list
```

Verificar PVCs restantes

Terminal input

```
kubectl get pvc -n <NAMESPACE>
```

Verificar PVs restantes

Terminal input

```
kubectl get pv | grep <NAMESPACE>
```

© Copyright Tecnisys 2026. All rights reserved.

Resumo

Situação	Caminho principal	Observação
Componente instalado diretamente via Helm	`helm uninstall <RELEASE_NAME> -n <NAMESPACE>`	Use quando houver release Helm real
Componente gerenciado pelo ArgoCD	`argocd app delete <componente> --cascade`	Remove a Application e os recursos gerenciados
Remoção completa do ambiente gerenciado pelo ArgoCD	`argocd app delete ... --cascade` em ordem reversa, ou remoção da app raiz	PVCs e PVs podem exigir limpeza manual
Remoção do próprio ArgoCD	`helm uninstall tdp-argo -n <NAMESPACE>`	Deve ocorrer depois da remoção das Applications
Remoção completa do ambiente instalado via Helm	sequência de `helm uninstall` em ordem reversa	Fluxo separado do modelo ArgoCD
Limpeza de dados persistentes	`kubectl delete pvc ...` e, se necessário, `kubectl delete pv ...`	Etapa manual e destrutiva
Limpeza do namespace	`kubectl delete namespace <NAMESPACE>`	Só quando o namespace for dedicado ao ambiente removido

© Copyright Tecnisys 2026. All rights reserved.

PARTE IV - EXTENSÕES

© Copyright Tecnisys 2026. All rights reserved.

Extensões

A diversidade de componentes do TDP (Tecnisys Data Platform) fomenta o surgimento de extensões para os mais diversas segmentos, fornecendo soluções que exploram o melhor da plataforma, sem comprometer sua estabilidade.

As extensões do TDP não são fornecidas por padrão na instalação da plataforma e devem ser contratadas separadamente.

Atualmente, está disponível para contratação a extensão TDP APIM Insights, a qual, através de componentes da plataforma TDP, possibilita a análise dos dados de uso de APIs gerenciadas por API Managers. Para mais informações, acesse o link abaixo:

- *TDP WSO2 APIM Insights*

Deseja contratar a extensão TDP APIM Insights? Entre em contato com a Tecnisys através do Fale Conosco.

© Copyright Tecnisys 2026. All rights reserved.

Extensão TDP APIM Insights

A extensão TDP APIM Insights permite a análise e visualização dos dados de uso das APIs gerenciadas por API Managers.

Após a instalação da extensão, os metadados das APIs são enviados automaticamente para a camada de barramento de dados do cluster TDP. Estes dados são então processados e armazenados em uma camada analítica, sendo apresentados em gráficos dinâmicos dentro de um painel interativo.

Dessa forma, os usuários podem acompanhar facilmente o uso das APIs, entender o comportamento dos serviços e aplicações, identificar problemas, além de tomar decisões mais assertivas, baseadas em dados.

Atualmente, a extensão TDP APIM Insights integra-se somente com WSO2 API Manager. Logo mais, novos API Managers serão suportados, ampliando ainda mais as possibilidades de uso da extensão.

Para saber mais sobre o processo de instalação da extensão, acesse o link abaixo:

- *Instalação da Extensão TDP APIM Insights para WSO2 API Manager*.

© Copyright Tecnisys 2026. All rights reserved.

Instalação do add-on para o WSO2

A extensão TDP WSO2 APIM Insights permite a análise e visualização dos dados de uso das APIs gerenciadas pelo WSO2.

Para sua instalação é necessário possuir o respectivo arquivo da extensão (.taf), fornecido individualmente pela Tecnisys após a contratação do produto. Se deseja adiquiri-lá, Entre em contato conosco.

Pré-Requisitos

- Arquivo da extensão TDP WSO2 APIM Insights (.taf).
- O *Cluster* TDP deve estar instalado e configurado, com os serviços do Apache NiFi, Apache Kafka, Apache Druid e Apache Superset em execução.
- O WSO2 API Manager deve estar instalado e configurado.
- O tdpctl, a Interface de Linha de Comando do TDP, deve estar instalada.

Sintaxe do Comando

Terminal input

```
tdpctl addon-install [Flags] /caminho/para/arquivo-extensão.taf
```

NOTE

Para informar o arquivo da extensão TDP (.taf) no comando, utilize a opção -f ou --file. Caso não informe o arquivo no comando, o tdpctl solicitará essa informação durante a execução. O caminho do arquivo deve ser absoluto.

© Copyright Tecnisys 2026. All rights reserved.

Para informações sobre o comando *addon-install*, informe a opção --help:

Terminal input

```
tdpctl addon-install --help
```

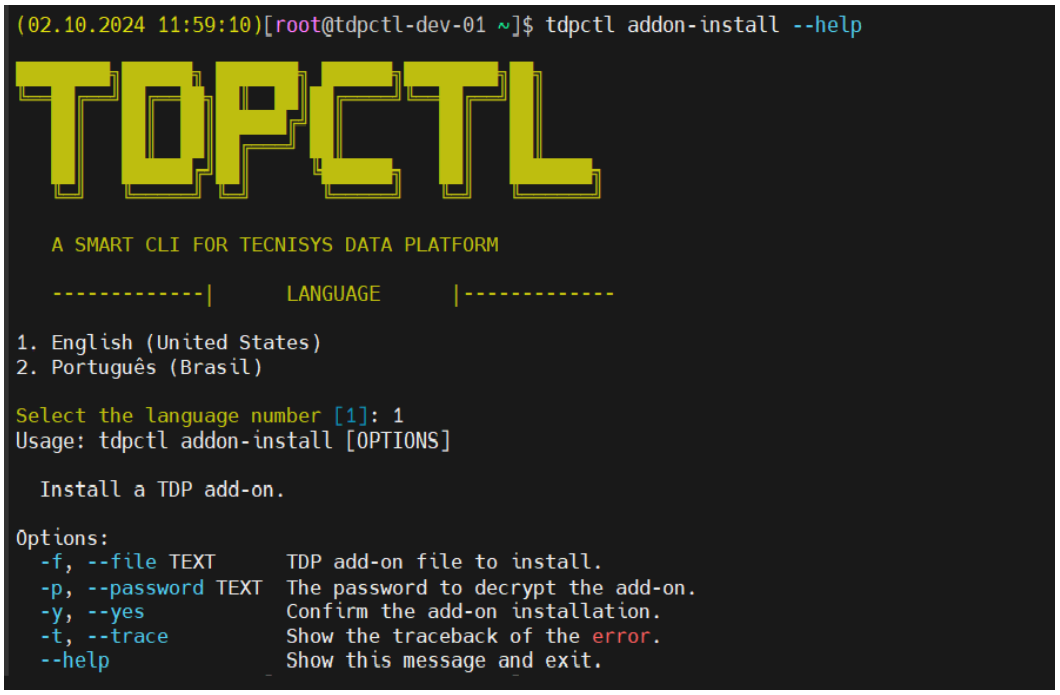


Figura 1 - Opção --help do comando tdpctl addon-install

Estágios da Instalação

A instalação da extensão TDP WSO2 APIM Insights deve ser conduzida em dois estágios, executados em sequência.

O primeiro, o estágio BIG-DATA, deve ser executado em um dos hosts do Cluster TDP.

Já o segundo, o estágio API-MANAGER, deve ser executado no host em que o WSO2 API Manager está instalado.

> **NOTE**
>
> No caso de uma implantação do WSO2 API Manager com mais de um nó, o estágio API-MANAGER deve ser executado em todas as máquinas.

© Copyright Tecnisys 2026. All rights reserved.

Para ambos os estágios, o comando de instalação é o mesmo, conforme sintaxe apresentada acima.

Estágio BigData

No estágio BIG-DATA, são requisitadas informações, realizadas verificações e executadas operações nos serviços do Cluster TDP utilizados pela extensão. São eles: Apache NiFi, Apache Kafka, Apache Druid e Apache Superset.

Instruções

Terminal input

```
tdpctl addon-install -f /caminho/para/arquivo-extensão.taf
```

No tdpctl, após a execução do comando de instalação da extensão (addon-install):

1. Selecione o idioma desejado (1-Inglês 2-Portugues).
2. Informe o arquivo da extensão TDP (.taf), caso este não tenha sido informado no comando (opção -f ou --file).
3. Informe a senha para decriptografar o arquivo.
4. Confirme que deseja iniciar a operação de instalação da extensão.
5. Selecione o estágio big-data (opção 1).

Na sequência, para cada serviço do Cluster TDP:

Ambari

6. Informe o host do Ambari Web [Default: localhost].
7. Informe a porta do Ambari Web [Default: 8080].
8. Informe o protocolo de comunicação utilizado pelo Ambari Web [Default: http].
9. Informe o nome do Cluster TDP definido no Ambari.
10. Informe o usuário de autenticação do Ambari [Default: admin].
11. Informe a senha de autenticação do Ambari.

NiFi

© Copyright Tecnisys 2026. All rights reserved.

12. Informe o usuário de autenticação do NiFi.
13. Informe a senha de autenticação do NiFi.

Druid

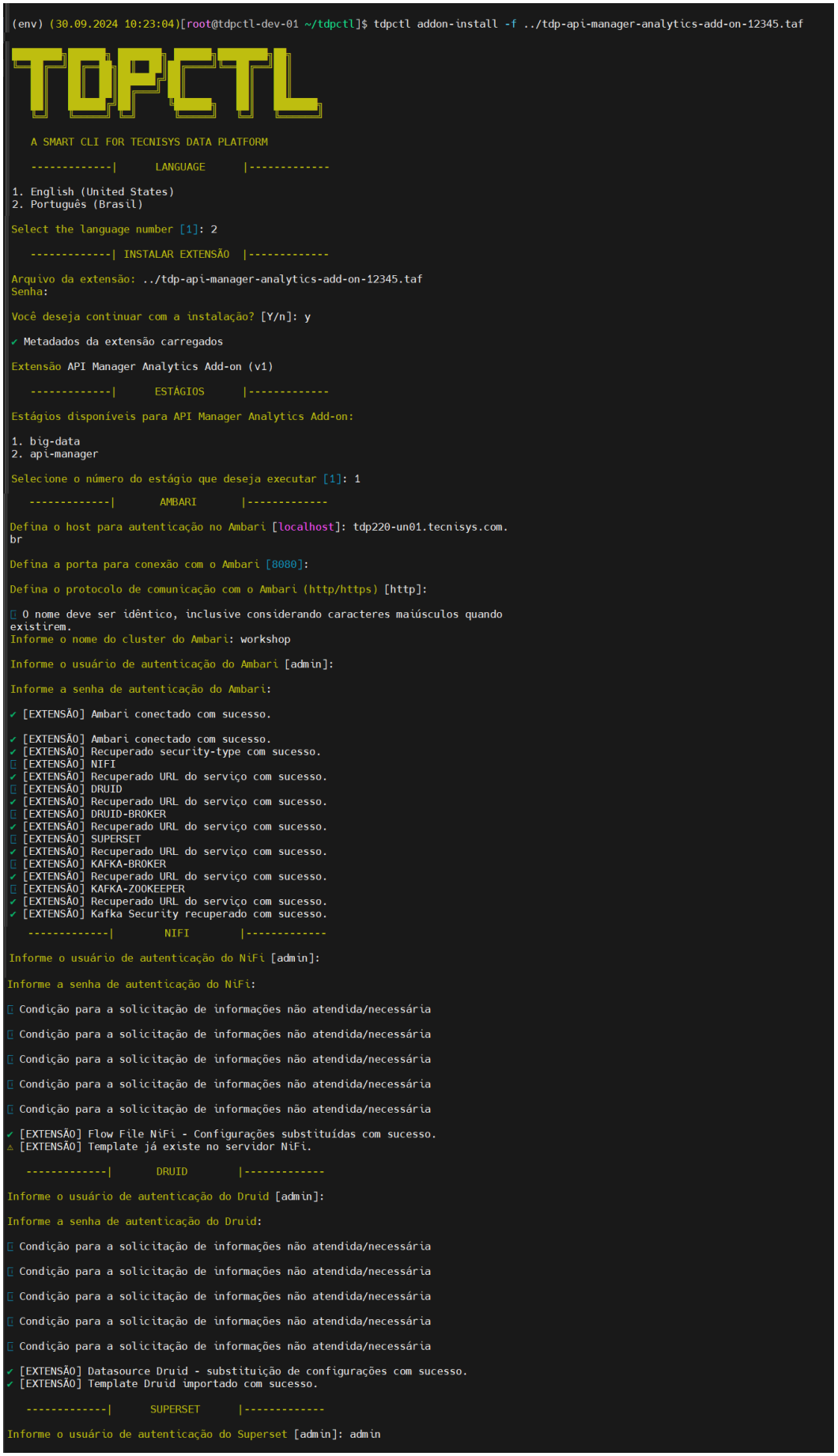
14. Informe o usuário de autenticação do Druid.
15. Informe a senha de autenticação do Druid.

Superset

16. Informe o usuário de autenticação do Superset.
17. Informe a senha de autenticação do Superset.

Confira todas as etapas do estágio BIG-DATA na imagem a seguir.

© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.

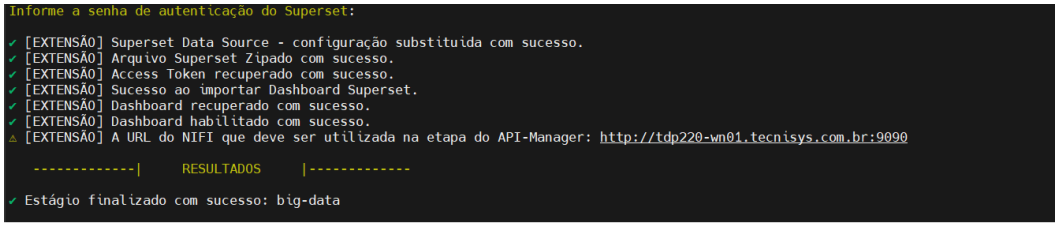


Figure 2. Execução do Estágio BIG-DATA

> **IMPORTANT**
>
> Ao final deste estágio é fornecida a URL do serviço do Apache NiFi. Lembre-se de copiá-la, pois será necessário informá-la no estágio API-MANAGER.

> **NOTE**
>
> Os logs da instalação da extensão TDP WSO2 APIM Insights são armazenados no diretório de instalação do tdpctl.
>
> Para visualizar os logs, execute o comando a seguir:
>
> Terminal input
>
> ```
> tail -f <DIRETORIO_tdpctl>/tdpctl.log
> ```
>
> A variável "DIRETORIO_tdpctl" corresponde ao diretório de instalação do tdpctl.

Estágio API-Manager

No estágio API-MANAGER, o WSO2 API Manager será configurado para enviar dados para o Cluster TDP, viabilizando, desta forma, o monitoramento das APIs.

Instruções

Terminal input

```
tdpctl addon-install -f /caminho/para/arquivo-extensão.taf
```

© Copyright Tecnisys 2026. All rights reserved.

No tdpctl, após a execução do comando de instalação da extensão (addon-install):

1. Selecione o idioma desejado. (1-Inglês 2-Portugues).
2. Informe o arquivo da extensão TDP (.taf), caso este não tenha sido informado no comando (opção -f ou --file).
3. Informe a senha para decriptografar o arquivo.
4. Confirme que deseja iniciar a operação de instalação da extensão.
5. Selecione o estágio api-manager (opção 2).

Na sequência, para o WSO2 API Manager, o usuário deve:

WSO2AM

6. Informe a URL do WSO2 API Manager para verificação da versão.
7. Informe o usuário do WSO2 API Manager [Default: admin]
8. Informe a senha do WSO2 API Manager.
9. Informe o caminho de instalação do WSO2 API Manager. [Default: /opt/wso2am]
10. Informe o nome do daemon service do WSO2 API Manager. [Default: wso2am]
11. Informe o usuario do serviço WSO2 API Manager. [Default: wso2user]
12. Informe a URL do serviço do Apache NiFi. Se o NiFi estiver em cluster, Informe a URL de um dos nós.

Confira todas as etapas do estágio API-MANAGER na imagem a seguir.

© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.



Figura 3 - Execução do Estágio API-MANAGER

> **NOTE**
>
> Os logs da instalação da extensão TDP WSO2 APIM Insights são armazenados no diretório de instalação do tdpctl.
>
> Para visualizar os logs, execute o comando a seguir:
>
> **Terminal input**
>
> ```
> tail -f <DIRETORIO_TDPCTL>/tdpctl.log
> ```
>
> A variável "DIRETORIO_tdpctl" corresponde ao diretório de instalação do tdpctl.

Análise e Visualização dos Dados

Após a instalação da extensão TDP WSO2 APIM Insights, os dados de uso das APIs gerenciadas pelo WSO2 API Manager podem ser analisados e visualizados no Apache Superset.

Instruções

1. Acesse a interface web do Apache Superset. Link disponível na seção Quick Links do Ambari Web.

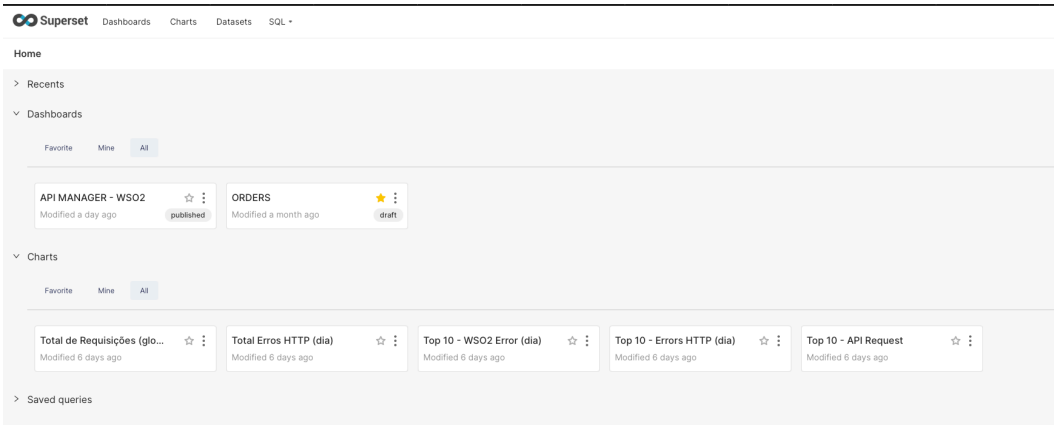


Figura 4 - Página de início do Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

2. Na aba "Dashboards", Selecione o painel API MANAGER-WSO2.

Seleção do item API MANAGER-WSO2

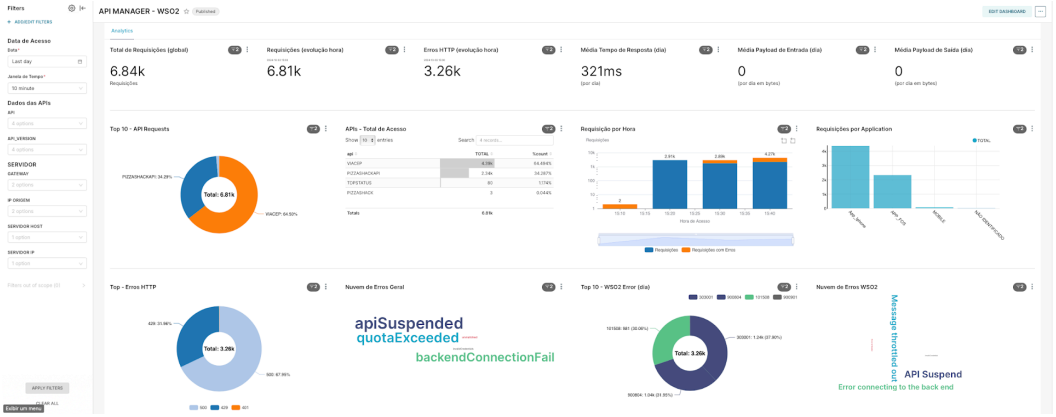


Figura 5 - Visualização do painel API MANAGER - WSO2

3. Utilize os filtros e tabelas disponíveis para analisar os dados de uso das APIs.

Visualização no Superset - API MANAGER-WSO2

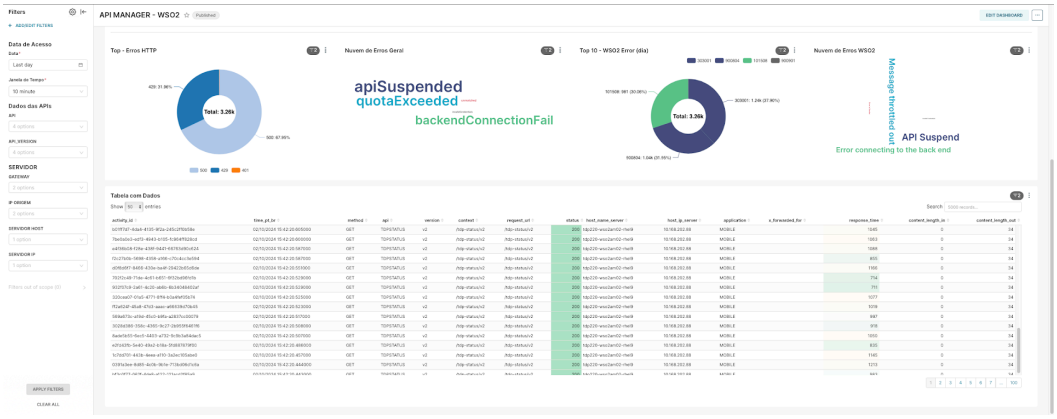


Figura 6 - Visualização no Superset - API MANAGER-WSO2

© Copyright Tecnisys 2026. All rights reserved.

PARTE V - COMPONENTES

© Copyright Tecnisys 2026. All rights reserved.

Matriz de Versões

As informações sobre versões dos componentes do TDP estão organizadas por edição da plataforma.

Para consultar a matriz correspondente, acesse:

- Componentes TDP Datacenter
- Componentes TDP Kubernetes

Cada página reúne a matriz de versões oficial dos componentes disponíveis para a respectiva edição do TDP.

© Copyright Tecnisys 2026. All rights reserved.

Componentes TDP Datacenter

Matriz de Versão oficial dos componentes disponíveis na edição **TDP Datacenter** 3.0.0:

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Airflow	2.10.5
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ambari	3.0.0.4
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Ambari Infra Solr	3.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Ambari Metrics	3.1.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Atlas	2.3.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Delta Lake	3.3.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Druid	32.0.0

© Copyright Tecnisys 2026. All rights reserved.

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Flink	1.17.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Grafana	11.5.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Great Expectations	1.3.5
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Hadoop	3.3.6
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache HBase	2.5.7
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	HDFS	3.3.6
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Hive	4.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	HiveLlap	4.0.0

© Copyright Tecnisys 2026. All rights reserved.

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Hive Metastore	4.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Iceberg	1.8.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	JupyterLab	4.4.5
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Kafka	3.4.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Kerberos	1.10.3-30
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Knox	2.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Livy	0.8.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	MapReduce	3.3.6

© Copyright Tecnisys 2026. All rights reserved.

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi	1.28.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi Registry	1.28.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	OpenMetadata	1.9.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ozone	1.4.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Phoenix	5.2.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ranger	2.4.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Ranger KMS	2.4.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Solr	8.11.1

© Copyright Tecnisys 2026. All rights reserved.

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Spark	3.5.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Sqoop *(obsoleto — apo-sentado pela Apa-che Software Foundation, remo-ção prevista no TDP 3.1.0)*	1.4.7
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Superset	4.1.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Tez	0.10.4
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Trino	467
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Yarn	3.3.6
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zeppelin	0.11.2

© Copyright Tecnisys 2026. All rights reserved.

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zookeeper	3.8.4

Consulte os requisítos mínimos para a instalação do TDP Datacenter aqui.

© Copyright Tecnisys 2026. All rights reserved.

Componentes TDP Kubernetes

Matriz de Versão oficial dos componentes disponíveis na edição **TDP Kubernetes** 3.0.1:

Componente	Versão Compo-nente	Helm Chart	Versão Chart	Orquestrador	Versão
Apache Airflow	3.0.2	`tdp-airflow`	3.0.1	Kubernetes	1.32+
Apache Airflow	3.0.2	`tdp-airflow`	3.0.1	Red Hat OpenShift	4.19+
Apache Airflow	3.0.2	`tdp-airflow`	3.0.1	Rancher Manager	2.10.x+
ArgoCD	3.2.5	`tdp-argo`	3.0.1	Kubernetes	1.32+
ArgoCD	3.2.5	`tdp-argo`	3.0.1	Red Hat OpenShift	4.19+
ArgoCD	3.2.5	`tdp-argo`	3.0.1	Rancher Manager	2.10.x+
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.1	Kubernetes	1.32+
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.1	Red Hat OpenShift	4.19+
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.1	Rancher Manager	2.10.x+

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Compo-nente	Helm Chart	Versão Chart	Orquestrador	Versão
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.1	Kubernetes	1.32+
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.1	Red Hat OpenShift	4.19+
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.1	Rancher Manager	2.10.x+
CRDs Bundle	3.0.1	`tdp-crds`	3.0.1	Kubernetes	1.32+
CRDs Bundle	3.0.1	`tdp-crds`	3.0.1	Red Hat OpenShift	4.19+
CRDs Bundle	3.0.1	`tdp-crds`	3.0.1	Rancher Manager	2.10.x+
Delta Lake	4.0.1	`tdp-deltalake`	3.0.1	Kubernetes	1.32+
Delta Lake	4.0.1	`tdp-deltalake`	3.0.1	Red Hat OpenShift	4.19+
Delta Lake	4.0.1	`tdp-deltalake`	3.0.1	Rancher Manager	2.10.x+
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.1	Kubernetes	1.32+
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.1	Red Hat OpenShift	4.19+

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Compo-nente	Helm Chart	Versão Chart	Orquestrador	Versão
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.1	Rancher Manager	2.10.x+
Apache Iceberg	1.10.0	`tdp-iceberg`	3.0.1	Kubernetes	1.32+
Apache Iceberg	1.10.0	`tdp-iceberg`	3.0.1	Red Hat OpenShift	4.19+
Apache Iceberg	1.10.0	`tdp-iceberg`	3.0.1	Rancher Manager	2.10.x+
JupyterHub	5.3.0	`tdp-jupyter`	3.0.1	Kubernetes	1.32+
JupyterHub	5.3.0	`tdp-jupyter`	3.0.1	Red Hat OpenShift	4.19+
JupyterHub	5.3.0	`tdp-jupyter`	3.0.1	Rancher Manager	2.10.x+
Apache Kafka	4.1.0	`tdp-kafka`	3.0.1	Kubernetes	1.32+
Apache Kafka	4.1.0	`tdp-kafka`	3.0.1	Red Hat OpenShift	4.19+
Apache Kafka	4.1.0	`tdp-kafka`	3.0.1	Rancher Manager	2.10.x+
Apache NiFi	1.28.0	`tdp-nifi`	3.0.1	Kubernetes	1.32+
Apache NiFi	1.28.0	`tdp-nifi`	3.0.1	Red Hat OpenShift	4.19+

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Componente	Helm Chart	Versão Chart	Orquestrador	Versão
Apache NiFi	1.28.0	`tdp-nifi`	3.0.1	Rancher Manager	2.10.x+
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.1	Kubernetes	1.32+
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.1	Red Hat OpenShift	4.19+
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.1	Rancher Manager	2.10.x+
Apache Ozone	2.0.0	`tdp-ozone`	3.0.1	Kubernetes	1.32+
Apache Ozone	2.0.0	`tdp-ozone`	3.0.1	Red Hat OpenShift	4.19+
Apache Ozone	2.0.0	`tdp-ozone`	3.0.1	Rancher Manager	2.10.x+
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.1	Kubernetes	1.32+
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.1	Red Hat OpenShift	4.19+
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.1	Rancher Manager	2.10.x+
Apache Ranger	2.7.0	`tdp-ranger`	3.0.1	Kubernetes	1.32+

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Compo-nente	Helm Chart	Versão Chart	Orquestrador	Versão
Apache Ranger	2.7.0	`tdp-ranger`	3.0.1	Red Hat OpenShift	4.19+
Apache Ranger	2.7.0	`tdp-ranger`	3.0.1	Rancher Manager	2.10.x+
Apache Spark	4.0.2	`tdp-spark`	3.0.1	Kubernetes	1.32+
Apache Spark	4.0.2	`tdp-spark`	3.0.1	Red Hat OpenShift	4.19+
Apache Spark	4.0.2	`tdp-spark`	3.0.1	Rancher Manager	2.10.x+
Apache Superset	5.0.0	`tdp-superset`	3.0.1	Kubernetes	1.32+
Apache Superset	5.0.0	`tdp-superset`	3.0.1	Red Hat OpenShift	4.19+
Apache Superset	5.0.0	`tdp-superset`	3.0.1	Rancher Manager	2.10.x+
Trino	478	`tdp-trino`	3.0.1	Kubernetes	1.32+
Trino	478	`tdp-trino`	3.0.1	Red Hat OpenShift	4.19+
Trino	478	`tdp-trino`	3.0.1	Rancher Manager	2.10.x+

© Copyright Tecnisys 2026. All rights reserved.



> **REGISTRO DE IMAGENS**
>
> Os Helm Charts do TDP Kubernetes estão disponíveis no registro OCI `registry.tecnisys.com.br/tdp/charts/`. Para obter as credenciais de acesso, entre em contato com a Tecnisys.

Consulte os pré-requisitos para a instalação do TDP Kubernetes aqui.

© Copyright Tecnisys 2026. All rights reserved.

PARTE VI - NOTAS TÉCNICAS

© Copyright Tecnisys 2026. All rights reserved.

Notas Técnicas

A seguir, apresentamos as notas técnicas do TDP 3.0. Nesta seção, você encontrará informações detalhadas sobre as correções e melhorias incorporadas nas releases da plataforma. Mantemos um registro claro e abrangente para garantir que você esteja sempre informado sobre as mudanças mais recentes.

- **Componentes Adicionados**: Confira os componentes adicionados
- **Componentes Removidos**: Confira os componentes removidos
- **Mais Informações**: Confira as notas técnicas de cada componente

© Copyright Tecnisys 2026. All rights reserved.

Componentes Adicionados

Os seguintes componentes foram adicionados no TDP 3.0

- JupyterLab
- OpenMetadata

© Copyright Tecnisys 2026. All rights reserved.

Componentes Removidos

Não foram removidos componentes nesta versão.

© Copyright Tecnisys 2026. All rights reserved.

DESTAQUES

© Copyright Tecnisys 2026. All rights reserved.

Destaques – TDP Datacenter

Plataforma

Versão	Destaques
TDP 3.0	**JupyterLab adicionado:** ambiente interativo para análise de dados, notebooks e visualizações; suporte a múltiplas linguagens e extensões; interface moderna baseada em abas, integrando edição de código, execução e gráficos.
	OpenMetadata adicionado: plataforma de governança e catálogo de dados; permite rastreabilidade (lineage), descoberta de dados e definição de qualidade; integração com diversos serviços do ecossistema (bancos, lakes, pipelines).

Tabela A. Destaques da plataforma TDP Datacenter

Componentes

Veja os destaques dos componentes atualizados:

Serviço	Versão	Categoria	Recurso/Destaque
Airflow	2.10.5	Orquestração	Tarefas de limpeza garantidas mesmo com falha ou sucesso manual — evita vaza-

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			mento de recursos.
			Resolução de diversos bugs, inclusive em trigger rules mapeadas, validação de IDs, XCom com "/", event logs e muito mais.
			Atualização de imagens do Chart Helm com versões mais recentes para Airflow, PgBouncer e exporters.
Delta Lake	3.3.0	Formato de tabela otimizado	**Identity Columns:** geração automática de valores únicos para novas linhas (Python e Scala) — modos `generatedAlwaysAs` e `generatedByDefaultAs`.
			VACUUM LITE: limpeza de arquivos obsoletos 5–10× mais rápida

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			usando apenas o log transacional.
			Row Tracking Backfill: habilita rastreamento de linha (row-level lineage) em tabelas existentes, incluindo metadados Row Id e Row Commit Version.
			Version Checksums: somas de verificação por commit, reforçando consistência, performance e debugging.
			UniForm ALTER: habilita o formato UniForm em tabelas existentes sem reescrever os dados.
			Type Widening: suporte a alargamento de tipos em Delta Kernel (Java e Rust).

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
Druid	32.0.0	Análise em Tempo Real	Foco em estabilidade e performance: mais de 341 commits de 52 colaboradores (~30% a mais que versões anteriores).
			Migração para SQL padrão (ANSI SQL compliance): configurações legadas de tratamento de null removidas — comportamento ANSI SQL fixo.
			Nova funcionalidade **Projection**: pré-cômputo de agregações para melhora significativa de performance em consultas.
Great Expectations	1.3.5	Qualidade de Dados	Nova funcionalidade: `CheckpointFactory.add_or_update` e suporte a `strict_min`/`strict_max` em `ExpectTabl`

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			`eRowCountToBeBetween`.
			Suporte a `row_condition` com datetimes em Pandas e Spark; agendamento de limpeza no BigQuery; parâmetro `strict` no tipo Window; garantia de `run_id` em `ValidationDefinition.run`.
HBase	2.5.7	Base de dados NoSQL	Switch para evitar reabertura de regiões ao editar tabelas — previne "storm" de RIT.
			Comando `isolate_regions` no RegionMover — isola e realoca regiões de forma precisa e controlada.
Hive	4.0.0	Exploração e Análise de Dados	Salto significativo: ~5 mil commits desde a versão 3.1.3.

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			Integração com Apache Iceberg: suporte aprimorado, incluindo compaction via `OPTIMIZE TABLE`.
			Transações ACID e locking aprimorados; compaction para tabelas ACID e Iceberg.
			Suporte a Docker: imagens oficiais no Docker Hub.
			Otimizações de execução: anti-join, branch pruning, histograma de colunas, HPL/SQL, scheduled queries, materialized views e Tez/LLAP.

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			Replicação aprimorada para tabelas externas e ACID; suporte ao Apache Ozone como armazenamento escalável.
			Recursos avançados: GeoSpatial nativo, Iceberg compaction, autenticação JWT no Metastore, SAML 2 no HiveServer2.
Iceberg	1.8.0	Table Formats	Fim do suporte ao Spark 3.3 e Hive Runtime.
			Deletion Vectors: nova spec, APIs e suporte na leitura/escrita.
			Novos tipos: Variant e UnknownType.
			Melhorias operacionais: fast append, remoção de specs não usados,

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			procedures úteis no Spark.
			Integração aprimorada com AWS/Azure; compatibilidade estendida com Spark, Flink e Hive.
Kafka	3.4.1	Streaming de Dados	Migração do ZooKeeper para KRaft (versão inicial; não recomendada para produção) — sem downtime.
			Novo campo `generation` no protocolo de consumo — auxilia gestão de reivindicações de partições.
			Snapshots de metadata baseados em tempo (ex.: a cada hora); consumidores rack-aware para AZ local.
			Kafka Streams (KIP-770 & KIP-

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			837): atualização de configs e métricas; broadcast de registros de saída para todas as partições.
			MirrorMaker2: lê todos os sincronizadores de offset ao iniciar; publica sincronizações durante commit; traduz offsets entre fluxos de replicação.
			Corrigido vazamento de recurso em interceptadores.
NiFi	1.28.1	Gestão de Fluxos de Dados	Corrigido registro de valores sensíveis de parâmetros em log de sincronização — não expostos mesmo com debug habilitado (1.28.1).
			Proteção contra XSS: descrições de parâmetros corre-

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			tamente neutralizadas (1.28.0).
			Fim do suporte para NiFi 1.x (encerrado em 08/12/2024) — recomendada migração para a série 2.x.
Ozone	1.4.1	Armazenamento de Objetos Escalável	Evita race condition no datanode ao criar VERSION — aumenta confiabilidade operacional.
			Logs do SCM refinados — reduz ruído e falsos erros ao lidar com sequence ID em containers fechados.
			Reforço de segurança no S3 Gateway — endpoint de segredos restrito a administradores.

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
Spark	3.5.3	Computação Distribuída	Terceira atualização de manutenção da série 3.5 — foco em segurança e correções de bugs. Recomendado para ambientes que já utilizam a série 3.5.
Superset	4.1.2	Visualização de Dados	Novos gráficos: Big Number com comparações temporais, Heatmap, Histograma e Sankey.
			Catálogo dinâmico em bancos de dados conectados; UI de upload mais intuitiva com validações.
			Filtros de tempo e macro Jinja para dinamismo em dashboards; melhorias de UX no SQL Lab e permissões.

© Copyright Tecnisys 2026. All rights reserved.

Serviço	Versão	Categoria	Recurso/Destaque
			Security fixes em 4.1.2: evita takeover de recursos e bypass de controle de acesso.
Apache Tez	0.10.4	Motor de Execução	~28 correções e melhorias — observabilidade, estabilidade e segurança.
ZooKeeper	3.8.4	Coordenação Distribuída	Limite configurável de conexões/clientes por servidor.
			zkCli: nova opção para aguardar conexão antes de executar comandos — evita falhas em ambientes instáveis.
			Logs de inicialização com unidade de tempo; endereços de rede mais legíveis; mensagens de erro aprimoradas.

Tabela B. Destaques dos componentes TDP Datacenter

© Copyright Tecnisys 2026. All rights reserved.

Destaques – TDP Kubernetes

> **NOVA EDIÇÃO — DISPONÍVEL A PARTIR DA VERSÃO 3.0**
>
> O **TDP Kubernetes** é uma nova edição da Tecnisys Data Platform, introduzida na versão 3.0. Oferece implantação cloud-native dos componentes de Big Data em clusters Kubernetes, com foco em portabilidade, escalabilidade e GitOps.
>
> **Principais características:**
>
> - Instalação e atualização via **Helm Charts** com registro OCI (`registry.tecnisys.com.br/tdp/charts/`)
> - Gestão declarativa via **ArgoCD** (GitOps)
> - Orquestradores certificados: **Kubernetes 1.32+**, **Red Hat OpenShift 4.19+** e **Rancher Manager 2.10.x+**
> - Mais de 18 componentes disponíveis desde o lançamento
>
> Para a lista completa de componentes e versões, consulte Componentes TDP Kubernetes.

Release	Destaque	Descrição
3.0.0	Lançamento da edição	Primeira edição cloud-native do TDP com 18+ componentes entregues via Helm Charts.
3.0.1	Homologação Red Hat OpenShift 4.19+	Suporte oficial ao Red Hat OpenShift 4.19+ com validação em clusters OpenShift e compatibilidade com Security Context Constraints (SCCs).

© Copyright Tecnisys 2026. All rights reserved.

Tabela C. Destaques do TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.

MAIS INFORMAÇÕES

© Copyright Tecnisys 2026. All rights reserved.

Mais Informações

Esta secção das notas técnicas está organizada por **edição** da Tecnisys Data Platform. Selecione a página da edição que corresponde ao seu ambiente:

Edição	Página	O que encontrar
TDP Datacenter	Mais Informações — TDP Datacenter	Tabela de componentes atualizados, resumo de issues (STACK / BIGTOP / AMBARI) e ligações à documentação e comunidades dos projetos upstream.
TDP Kubernetes	Mais Informações — TDP Kubernetes	Matriz Helm Chart / versão da aplicação, orquestradores homologados, componentes exclusivos e referências upstream de cada projeto.

Versões de referência desta documentação: TDP **3.0** — Datacenter **3.0.0**, Kubernetes **3.0.1**.

© Copyright Tecnisys 2026. All rights reserved.

Mais Informações – TDP Datacenter

Informações adicionais sobre os componentes da edição **TDP Datacenter** na versão 3.0.0.

Componentes Atualizados

A tabela abaixo apresenta os componentes que foram atualizados nesta versão do TDP Datacenter:

Componente	Versão Anterior	Versão Atual
Apache Airflow	2.9.3	2.10.5
Apache Ambari	3.0.0.3	3.0.0.4
Delta Lake	2.3.0	3.3.0
Apache Druid	30.0.0	32.0.0
Great Expectations	1.2.1	1.3.5
Apache HBase	2.5.6	2.5.7
Apache Hive	3.1.3	4.0.0
Apache Iceberg	1.5.0	1.8.0
Apache Kafka	2.8.2	3.4.1
Apache NiFi	1.27.0	1.28.1
Apache Spark	3.3.4	3.5.3
Apache Superset	4.0.2	4.1.2

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Anterior	Versão Atual
Apache Tez	0.10.2	0.10.4
Apache Zookeeper	3.7.2	3.8.4

Características da Edição Datacenter

A edição TDP Datacenter é destinada à implantação em datacenters on-premise, utilizando:

- **Apache Ambari** para gerenciamento e monitoramento do cluster
- Implantação em **bare-metal** ou **máquinas virtuais**
- Sistemas operacionais: CentOS, Red Hat, Rocky Linux, AlmaLinux (8.x e 9.x)
- Arquitetura x86-64

Para a matriz completa de componentes, consulte a página Componentes TDP Datacenter.

Resumo de Issues — TDP Datacenter 3.0.0

Seguem as `issues` resolvidas nesta versão:

Novos Componentes:

- STACK-8616 Adição do JupyterLab versão 4.4.5
- STACK-8615 Adição do OpenMetadata versão 1.9.1

Atualizações:

- BIGTOP-8615 Atualização do Airflow para a versão 2.10.5
- BIGTOP-8615 Atualização do Atlas para a versão 2.3.0
- BIGTOP-8615 Atualização do Ambari para a versão 3.0.0.4
- BIGTOP-8615 Atualização do Ambari Infra Solr para a versão 3.0.0.4
- BIGTOP-8615 Atualização do Ambari Metrics para a versão 3.1.0
- BIGTOP-8615 Atualização do Delta Lake para a versão 3.3.0
- BIGTOP-8615 Atualização do Druid para a versão 32.0.0
- BIGTOP-8615 Atualização do Flink para a versão 1.17.2

© Copyright Tecnisys 2026. All rights reserved.



- BIGTOP-8615 Atualização do Grafana para a versão 11.5.2
- BIGTOP-8615 Atualização do Great Expectations para a versão 1.3.5
- BIGTOP-8615 Atualização do Hadoop para a versão 3.3.6
- BIGTOP-8615 Atualização do HBase para a versão 2.5.7
- BIGTOP-8615 Atualização do HDFS para a versão 3.3.6
- BIGTOP-8615 Atualização do Hive para a versão 4.0.0
- BIGTOP-8615 Atualização do Hive LLAP para a versão 4.0.0
- BIGTOP-8615 Atualização do Hive Metastore para a versão 4.0.0
- BIGTOP-8615 Atualização do Iceberg para a versão 1.8.0
- BIGTOP-8615 Atualização do infra Solr para a versão 3.0.0
- BIGTOP-8615 Atualização do KafKa para a versão 3.4.1
- BIGTOP-8615 Atualização do Kerberos para a versão 1.10.3-30
- BIGTOP-8615 Atualização do Knox para a versão 2.0.0
- BIGTOP-8615 Atualização do Livy para a versão 0.8.0
- BIGTOP-8615 Atualização do NiFi para a versão 1.28.1
- BIGTOP-8615 Atualização do NiFi Registry para a versão 1.28.1
- BIGTOP-8615 Atualização do Ozone para a versão 1.4.1
- BIGTOP-8615 Atualização do Phoenix para a versão 5.2.0
- BIGTOP-8615 Atualização do Ranger para a versão 2.4.0
- BIGTOP-8615 Atualização do Ranger KMS para a versão 2.4.0
- BIGTOP-8615 Atualização do Solr para a versão 8.11.1
- BIGTOP-8615 Atualização do Sqoop para a versão 1.4.7 *(componente obsoleto — ver nota)*
- BIGTOP-8615 Atualização do Spark para a versão 3.5.3
- BIGTOP-8615 Atualização do Superset para a versão 4.1.2
- BIGTOP-8615 Atualização do Tez para a versão 0.10.4
- BIGTOP-8615 Atualização do Trino para a versão 467
- BIGTOP-8615 Atualização do Yarn para a versão 3.3.6
- BIGTOP-8615 Atualização do Zeppelin para a versão 0.11.2
- BIGTOP-8615 Atualização do ZooKeeper para a versão 3.8.4

Correções:

- STACK-8897 Interpretação de propriedades booleanas (props) no Airflow
- BIGTOP-8573 Problema com CLIs na instalação padrão do Airflow
- AMBARI-6468 Falha no Namenode ao identificar underline no hostname
- AMBARI-9530 Falha na compilação (build) do Ambari via Jenkins (Files View)

© Copyright Tecnisys 2026. All rights reserved.

- STACK-9346 Erro referente ao tipo de certificado no Ambari Infra Solr com SSL habilitado
- BIGTOP-9560 Erro no script de pós-instalação (after-INSTALL) do Ambari ao tentar configurar a versão do Atlas Server
- BIGTOP-9156 Falha no script 'import-hive'
- STACK-8282 Erro ao salvar no Iceberg usando Stream em tabela particionada
- AMBARI-METRICS-9218 Erro no método KafkaTimelineMetricsReporter presente no Ambari Metrics
- STACK-8724 Novo padrão para logs (Log4j2) no Knox
- BIGTOP-9549 Erro na inicialização da sessão Livy
- TDP-PATCHES-9290 Erro ao gravar protocolo OFS no Ozone
- STACK-9155 Funcionamento em HTTPS Solr (configuração das Keystores e Truststores)
- STACK-6314 Erro de sintaxe no Parâmetro DEFAULT_FEATURE_FLAGS do Superset ao carregar superset_config.py
- BIGTOP-8884 Erro na instalação do Trino devido à exigência do Python já instalada
- STACK-5655 Exception de Filesystem Closed ao iniciar o YARN RM
- AMBARI-9529 Ausência da biblioteca postgresql14-contrib no server de metadados

Melhorias:

- AMBARI-9312 Upgrade do Grafana e do Ambari Metrics
- STACK-8458 HBase Thrift Server como serviço
- STACK-8615 Rolling e Express Upgrade para o TDP 3.0.0

Documentação e Comunidades Upstream

Links para as notas técnicas, Jira e repositórios dos projetos que compõem o TDP Datacenter.

Apache Airflow 2.10.5

Notas Técnicas

GitHub

Apache Ambari 3.0.0.4

© Copyright Tecnisys 2026. All rights reserved.

- Notas Técnicas
- Jira
- GitHub

Apache Atlas 2.3.0

- Notas Técnicas
- Jira
- GitHub

Delta Lake 3.3.0

- Notas Técnicas
- GitHub

Apache Druid 32.0.0

- Notas Técnicas
- GitHub

Apache Flink 1.17.2

- Notas Técnicas
- Jira

© Copyright Tecnisys 2026. All rights reserved.

- GitHub

Grafana 11.5.2

- Notas Técnicas
- GitHub

Great Expectations 1.3.5

- Notas Técnicas
- GitHub

Apache Hadoop 3.3.6

- Notas Técnicas
- GitHub

Apache HBase 2.5.7

- Notas Técnicas
- GitHub

Apache Hive 4.0.0

- Notas Técnicas
- Jira

© Copyright Tecnisys 2026. All rights reserved.

- GitHub

Apache Iceberg 1.8.0

- Notas Técnicas
- GitHub

JupyterLab 4.4.5

- Notas Técnicas
- GitHub

Apache Kafka 3.4.1

- Notas Técnicas
- GitHub

Apache Kerberos 1.10.3-30

- Notas Técnicas
- GitHub

Apache Knox 2.0.0

- Notas Técnicas
- GitHub

Apache Livy 0.8.0

© Copyright Tecnisys 2026. All rights reserved.

- Notas Técnicas
- GitHub

Apache NiFi 1.28.1

- Notas Técnicas
- GitHub

OpenMetadata 1.9.1

- Notas Técnicas
- GitHub

Apache Ozone 1.4.1

- Notas Técnicas
- GitHub

Apache Phoenix 5.2.0

- Notas Técnicas
- GitHub

Apache Ranger 2.4.0

- Notas Técnicas

© Copyright Tecnisys 2026. All rights reserved.

- GitHub

Apache Solr 8.11.1

- Notas Técnicas
- GitHub

Apache Spark 3.5.3

- Notas Técnicas
- GitHub

Apache Sqoop 1.4.7

> **COMPONENTE OBSOLETO — REMOÇÃO PREVISTA NO TDP 3.1.0**
>
> O Apache Sqoop foi **aposentado pela Apache Software Foundation** e movido para o Apache Attic. Não recebe mais atualizações, correções de segurança ou suporte da comunidade. A última versão estável é a **1.4.7** (2017). **A partir do TDP 3.1.0, este componente será removido da plataforma.**

- Notas Técnicas
- GitHub

Apache Superset 4.1.2

- Notas Técnicas
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Tez 0.10.4

- Notas Técnicas
- GitHub

Trino 467

- Notas Técnicas
- GitHub

Apache Zeppelin 0.11.2

- Notas Técnicas
- GitHub

Apache ZooKeeper 3.8.4

- Notas Técnicas
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Mais Informações – TDP Kubernetes

Informações adicionais sobre os componentes da edição **TDP Kubernetes** na versão 3.0.1.

> **NOVA EDIÇÃO**
>
> O TDP Kubernetes é uma nova edição introduzida na versão 3.0, oferecendo implantação cloud-native via Helm Charts e ArgoCD.

Componentes TDP Kubernetes

Todos os componentes abaixo são entregues via Helm Chart (`tdp-*`) no registro OCI `registry.tecnisys.com.br/tdp/charts/`.

Componente	Versão	Helm Chart	Versão Chart
Apache Airflow	3.0.2	`tdp-airflow`	3.0.1
ArgoCD	3.2.5	`tdp-argo`	3.0.1
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.1
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.1
CRDs Bundle	3.0.1	`tdp-crds`	3.0.1
Delta Lake	4.0.1	`tdp-deltalake`	3.0.1
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.1
Apache Iceberg	1.10.0	`tdp-iceberg`	3.0.1

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão	Helm Chart	Versão Chart
JupyterHub	5.3.0	`tdp-jupyter`	3.0.1
Apache Kafka	4.1.0	`tdp-kafka`	3.0.1
Apache NiFi	1.28.0	`tdp-nifi`	3.0.1
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.1
Apache Ozone	2.0.0	`tdp-ozone`	3.0.1
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.1
Apache Ranger	2.7.0	`tdp-ranger`	3.0.1
Apache Spark	4.0.2	`tdp-spark`	3.0.1
Apache Superset	5.0.0	`tdp-superset`	3.0.1
Trino	478	`tdp-trino`	3.0.1

Componentes Exclusivos do TDP Kubernetes

Os seguintes componentes estão disponíveis exclusivamente na distribuição Kubernetes do TDP:

- **ClickHouse** — Banco de dados OLAP colunar de alta performance
- **CloudBeaver** — Ferramenta web de administração de bancos de dados

Características da Edição Kubernetes

A edição TDP Kubernetes é destinada à implantação em clusters Kubernetes, utilizando:

- **Helm Charts** para empacotamento e instalação dos componentes
- **ArgoCD** para GitOps e entrega contínua
- Registro OCI: `registry.tecnisys.com.br/tdp/charts/`

© Copyright Tecnisys 2026. All rights reserved.

- Orquestradores homologados: Kubernetes 1.32+, Red Hat OpenShift 4.19+, Rancher Manager 2.10.x+

Para a matriz completa de componentes, consulte a página Componentes TDP Kubernetes.

Documentação e Comunidades Upstream

As versões na matriz acima referem-se aos charts `tdp-*`. As ligações abaixo apontam para a documentação e repositórios dos projetos **upstream** correspondentes.

Argo CD 3.2.5

- Documentação
- GitHub

Apache Airflow 3.0.2

- Notas Técnicas
- GitHub

ClickHouse 25.8.11.66

- Changelog
- GitHub

CloudBeaver 25.3.0

- Releases

© Copyright Tecnisys 2026. All rights reserved.

- GitHub

Delta Lake 4.0.1

- Releases
- GitHub

Apache Iceberg 1.10.0

- Releases
- GitHub

JupyterHub 5.3.0

- Changelog
- GitHub

Apache Kafka 4.1.0 — Strimzi Operator

- Blog / Releases
- GitHub

Apache NiFi 1.28.0 — NiFiKop Operator

- Notas Técnicas NiFi
- GitHub NiFiKop

OpenMetadata 1.12.4

© Copyright Tecnisys 2026. All rights reserved.

- Releases
- GitHub

Apache Ozone 2.0.0

- Notas Técnicas
- GitHub

PostgreSQL 17.5.0

- Notas Técnicas
- GitHub

Apache Ranger 2.7.0

- Notas Técnicas
- GitHub

Apache Spark 4.0.2

- Notas Técnicas
- GitHub

Apache Superset 5.0.0

- Releases

© Copyright Tecnisys 2026. All rights reserved.

- GitHub

Trino 478

- Notas Técnicas
- GitHub

Hive Metastore 4.0.0

- Releases
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

NOTAS DE VERSÃO

© Copyright Tecnisys 2026. All rights reserved.

TDP Datacenter – Notas

© Copyright Tecnisys 2026. All rights reserved.

TDP Datacenter 3.0.0

Confira todas as funcionalidades, melhorias e correções implementadas no TDP Datacenter 3.0.0:

Novos Componentes e Funcionalidades

- **JupyterLab 4.4.5** — Ambiente interativo para análise de dados, notebooks e visualizações; suporte a múltiplas linguagens e extensões; interface moderna baseada em abas, integrando edição de código, execução e gráficos.
- **OpenMetadata 1.9.1** — Plataforma de governança e catálogo de dados; permite rastreabilidade (lineage), descoberta de dados e definição de qualidade; integração com diversos serviços do ecossistema (bancos, lakes, pipelines).

Melhorias

Componente	Versão Anterior	Versão Atual
Apache Airflow	2.9.3	2.10.5
Apache Ambari	3.0.0.3	3.0.0.4
Ambari Infra Solr	3.0.0	3.0.0
Ambari Metrics	3.1.0	3.1.0
Apache Atlas	2.3.0	2.3.0
Delta Lake	2.3.0	3.3.0
Apache Druid	30.0.0	32.0.0
Apache Flink	1.17.2	1.17.2
Grafana	11.5.2	11.5.2

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Anterior	Versão Atual
Great Expectations	1.2.1	1.3.5
Apache Hadoop	3.3.6	3.3.6
Apache HBase	2.5.6	2.5.7
Apache Hive	3.1.3	4.0.0
Apache Hive LLAP	—	4.0.0
Apache Hive Metastore	3.1.3	4.0.0
Apache Iceberg	1.5.0	1.8.0
Apache Kafka	2.8.2	3.4.1
Apache Knox	2.0.0	2.0.0
Apache Livy	0.8.0	0.8.0
Apache NiFi	1.27.0	1.28.1
Apache NiFi Registry	1.27.0	1.28.1
Apache Ozone	1.4.0	1.4.1
Apache Phoenix	5.2.0	5.2.0
Apache Ranger	2.4.0	2.4.0
Apache Ranger KMS	2.4.0	2.4.0
Apache Solr	8.11.1	8.11.1

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão Anterior	Versão Atual
Apache Spark	3.3.4	3.5.3
Apache Sqoop	1.4.7	1.4.7
Apache Superset	4.0.2	4.1.2
Apache Tez	0.10.2	0.10.4
Trino	467	467
Apache Yarn	3.3.6	3.3.6
Apache Zeppelin	0.11.2	0.11.2
Apache ZooKeeper	3.7.2	3.8.4

Issues Resolvidas

Novos Componentes:

- STACK-8616 Adição do JupyterLab versão 4.4.5
- STACK-8615 Adição do OpenMetadata versão 1.9.1

Atualizações:

- BIGTOP-8615 Atualização do Airflow para a versão 2.10.5
- BIGTOP-8615 Atualização do Atlas para a versão 2.3.0
- BIGTOP-8615 Atualização do Ambari para a versão 3.0.0.4
- BIGTOP-8615 Atualização do Ambari Infra Solr para a versão 3.0.0.4
- BIGTOP-8615 Atualização do Ambari Metrics para a versão 3.1.0
- BIGTOP-8615 Atualização do Delta Lake para a versão 3.3.0
- BIGTOP-8615 Atualização do Druid para a versão 32.0.0
- BIGTOP-8615 Atualização do Flink para a versão 1.17.2
- BIGTOP-8615 Atualização do Grafana para a versão 11.5.2
- BIGTOP-8615 Atualização do Great Expectations para a versão 1.3.5
- BIGTOP-8615 Atualização do Hadoop para a versão 3.3.6

© Copyright Tecnisys 2026. All rights reserved.



- BIGTOP-8615 Atualização do HBase para a versão 2.5.7
- BIGTOP-8615 Atualização do HDFS para a versão 3.3.6
- BIGTOP-8615 Atualização do Hive para a versão 4.0.0
- BIGTOP-8615 Atualização do Hive LLAP para a versão 4.0.0
- BIGTOP-8615 Atualização do Hive Metastore para a versão 4.0.0
- BIGTOP-8615 Atualização do Iceberg para a versão 1.8.0
- BIGTOP-8615 Atualização do Ambari Infra Solr para a versão 3.0.0
- BIGTOP-8615 Atualização do Kafka para a versão 3.4.1
- BIGTOP-8615 Atualização do Kerberos para a versão 1.10.3-30
- BIGTOP-8615 Atualização do Knox para a versão 2.0.0
- BIGTOP-8615 Atualização do Livy para a versão 0.8.0
- BIGTOP-8615 Atualização do NiFi para a versão 1.28.1
- BIGTOP-8615 Atualização do NiFi Registry para a versão 1.28.1
- BIGTOP-8615 Atualização do Ozone para a versão 1.4.1
- BIGTOP-8615 Atualização do Phoenix para a versão 5.2.0
- BIGTOP-8615 Atualização do Ranger para a versão 2.4.0
- BIGTOP-8615 Atualização do Ranger KMS para a versão 2.4.0
- BIGTOP-8615 Atualização do Solr para a versão 8.11.1
- BIGTOP-8615 Atualização do Sqoop para a versão 1.4.7 *(componente obsoleto — remoção prevista no TDP 3.1.0)*
- BIGTOP-8615 Atualização do Spark para a versão 3.5.3
- BIGTOP-8615 Atualização do Superset para a versão 4.1.2
- BIGTOP-8615 Atualização do Tez para a versão 0.10.4
- BIGTOP-8615 Atualização do Trino para a versão 467
- BIGTOP-8615 Atualização do Yarn para a versão 3.3.6
- BIGTOP-8615 Atualização do Zeppelin para a versão 0.11.2
- BIGTOP-8615 Atualização do ZooKeeper para a versão 3.8.4

Correções:

- STACK-8897 Interpretação de propriedades booleanas (props) no Airflow
- BIGTOP-8573 Problema com CLIs na instalação padrão do Airflow
- AMBARI-6468 Falha no Namenode ao identificar underline no hostname
- AMBARI-9530 Falha na compilação (build) do Ambari via Jenkins (Files View)
- STACK-9346 Erro referente ao tipo de certificado no Ambari Infra Solr com SSL habilitado
- BIGTOP-9560 Erro no script de pós-instalação (after-INSTALL) do Ambari ao tentar configurar a versão do Atlas Server

© Copyright Tecnisys 2026. All rights reserved.



- BIGTOP-9156 Falha no script 'import-hive'
- STACK-8282 Erro ao salvar no Iceberg usando Stream em tabela particionada
- AMBARI-METRICS-9218 Erro no método KafkaTimelineMetricsReporter presente no Ambari Metrics
- STACK-8724 Novo padrão para logs (Log4j2) no Knox
- BIGTOP-9549 Erro na inicialização da sessão Livy
- TDP-PATCHES-9290 Erro ao gravar protocolo OFS no Ozone
- STACK-9155 Funcionamento em HTTPS Solr (configuração das Keystores e Truststores)
- STACK-6314 Erro de sintaxe no Parâmetro DEFAULT_FEATURE_FLAGS do Superset ao carregar superset_config.py
- BIGTOP-8884 Erro na instalação do Trino devido à exigência do Python já instalada
- STACK-5655 Exception de Filesystem Closed ao iniciar o YARN RM
- AMBARI-9529 Ausência da biblioteca postgresql14-contrib no server de metadados

Melhorias:

- AMBARI-9312 Upgrade do Grafana e do Ambari Metrics
- STACK-8458 HBase Thrift Server como serviço
- STACK-8615 Rolling e Express Upgrade para o TDP 3.0.0

© Copyright Tecnisys 2026. All rights reserved.



TDP Kubernetes – Notas

© Copyright Tecnisys 2026. All rights reserved.

TDP Kubernetes 3.0.0

Confira todas as funcionalidades e componentes disponíveis no lançamento inicial da edição **TDP Kubernetes 3.0.0**:

> **NOVA EDIÇÃO**
>
> O TDP Kubernetes é uma nova edição introduzida no TDP 3.0, oferecendo implantação cloud-native via Helm Charts e ArgoCD em clusters Kubernetes 1.27+.

Novos Componentes e Funcionalidades

Todos os componentes abaixo são novos nesta edição. Cada componente é empacotado como um Helm Chart independente e gerenciado via ArgoCD.

Componente	Versão	Helm Chart	Versão Chart
Apache Airflow	3.0.2	`tdp-airflow`	3.0.0
ArgoCD	3.2.5	`tdp-argo`	3.0.0
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.0
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.0
CRDs Bundle	3.0.0	`tdp-crds`	3.0.0
Delta Lake	4.0.0	`tdp-deltalake`	3.0.0
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.0
Apache Iceberg	1.4.2	`tdp-iceberg`	3.0.0

© Copyright Tecnisys 2026. All rights reserved.



Componente	Versão	Helm Chart	Versão Chart
JupyterHub	4.5.6	`tdp-jupyter`	3.0.0
Apache Kafka	4.1.0	`tdp-kafka`	3.0.0
Apache NiFi	1.28.0	`tdp-nifi`	3.0.0
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.0
Apache Ozone	2.0.0	`tdp-ozone`	3.0.0
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.0
Apache Ranger	2.7.0	`tdp-ranger`	3.0.0
Apache Spark	4.0.0	`tdp-spark`	3.0.0
Apache Superset	5.0.0	`tdp-superset`	3.0.0
Trino	478	`tdp-trino`	3.0.0

Componentes Exclusivos do TDP Kubernetes

Os seguintes componentes estão disponíveis **exclusivamente** na edição Kubernetes:

- **ClickHouse** — Banco de dados OLAP colunar de alta performance
- **CloudBeaver** — Ferramenta web de administração de bancos de dados

Características da Edição

- **Helm Charts** para empacotamento e instalação dos componentes
- **ArgoCD** para GitOps e entrega contínua
- Registro OCI: `registry.tecnisys.com.br`
- Kubernetes 1.27+

© Copyright Tecnisys 2026. All rights reserved.

TDP Kubernetes 3.0.1

Confira todas as atualizações e correções implementadas no **TDP Kubernetes 3.0.1**:

Melhorias

Homologação para Red Hat OpenShift 4.19+

O TDP Kubernetes 3.0.1 amplia o suporte oficial a orquestradores com a **homologação para Red Hat OpenShift 4.19+**.

A partir desta versão, todos os charts Helm do TDP Kubernetes foram validados em clusters OpenShift, garantindo compatibilidade com os mecanismos de segurança reforçados dessa plataforma, como Security Context Constraints (SCCs), políticas de rede e gestão de identidade integrada.

Equipes que operam ambientes corporativos baseados em OpenShift podem agora implantar e gerenciar o stack TDP com o mesmo conjunto de charts, valores e procedimentos documentados para Kubernetes upstream, beneficiando-se da governança, do suporte enterprise e do ecossistema Red Hat.

Outras melhorias e correções

Esta versão também inclui ajustes de qualidade, refinamentos na documentação técnica e correções pontuais identificadas no ciclo de uso do 3.0.0:

- Correções de comportamento em casos de borda na configuração de Ingress e Gateway API.
- Refinamentos nos parâmetros de segurança e na geração de Secrets em charts selecionados.
- Melhorias de estabilidade e consistência em procedimentos de atualização via Helm e ArgoCD.
- Ajustes na documentação de integração e segurança dos componentes, com maior cobertura de exemplos e orientações operacionais.

Componentes TDP Kubernetes 3.0.1

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão	Helm Chart	Versão Chart
Apache Airflow	3.0.2	`tdp-airflow`	3.0.1
ArgoCD	3.2.5	`tdp-argo`	3.0.1
ClickHouse	25.8.11.66	`tdp-clickhouse`	3.0.1
CloudBeaver	25.3.0	`tdp-cloudbeaver`	3.0.1
CRDs Bundle	3.0.1	`tdp-crds`	3.0.1
Delta Lake	4.0.1	`tdp-deltalake`	3.0.1
Hive Metastore	4.0.0	`tdp-hive-metastore`	3.0.1
Apache Iceberg	1.10.0	`tdp-iceberg`	3.0.1
JupyterHub	5.3.0	`tdp-jupyter`	3.0.1
Apache Kafka	4.1.0	`tdp-kafka`	3.0.1
Apache NiFi	1.28.0	`tdp-nifi`	3.0.1
OpenMetadata	1.12.4	`tdp-openmetadata`	3.0.1
Apache Ozone	2.0.0	`tdp-ozone`	3.0.1
PostgreSQL	17.5.0	`tdp-postgresql`	3.0.1
Apache Ranger	2.7.0	`tdp-ranger`	3.0.1
Apache Spark	4.0.2	`tdp-spark`	3.0.1
Apache Superset	5.0.0	`tdp-superset`	3.0.1

© Copyright Tecnisys 2026. All rights reserved.

Componente	Versão	Helm Chart	Versão Chart
Trino	478	`tdp-trino`	3.0.1

Issues Resolvidas

Nenhuma issue pública registrada neste patch.

© Copyright Tecnisys 2026. All rights reserved.

CORREÇÕES

© Copyright Tecnisys 2026. All rights reserved.

Correções – TDP Datacenter

Não foram registradas correções específicas da edição **TDP Datacenter** nesta versão.

© Copyright Tecnisys 2026. All rights reserved.

Correções – TDP Kubernetes

Não foram registradas correções específicas da edição **TDP Kubernetes** nesta versão.

© Copyright Tecnisys 2026. All rights reserved.

PARTE VII - ROADMAP

© Copyright Tecnisys 2026. All rights reserved.

Roadmap

Neste tópico apresentamos os destaques das versões do TDP, incluindo as novidades previstas para os próximos semestres.

Versão Atual

TDP 3.0 | 2025-S1

- **Adição do Jupyter:** Serviço adicional de notebook interativo.
- **Adição do OpenMetadata:** Serviço de gerenciamento unificado de metadados.
- **Adição do ClickHouse** [1]**:** Serviço de banco de dados OLAP para análise de dados em tempo real.
- **TDP Kubernetes:** Edição Cloud Native.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

Versões Previstas

TDP 3.1 | 2026-S2

- **Adição do HUE:** Interface web para exploração, consulta e gerenciamento de dados.
- **Adição do MLflow:** Serviço para governança do ciclo de vida de modelos de AI/LLM.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

TDP 3.2 | 2027-S1

- **Adição do Milvus:** Serviço de banco de dados vetorial de alto desempenho para aplicações de AI.
- **Adição do Agno:** Framework para construção de sistemas multi-agentes.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

© Copyright Tecnisys 2026. All rights reserved.

TDP 4.0 | 2027-S2

- **Tecnisys Data eXperience:** Camada *low code* para a construção de projetos de BI, Data Science e IA.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

Versões Anteriores

TDP 2.3 | 2025-S1

- **Adição do Trino:** Serviço de virtualização de dados.
- **Adição do Apache Ozone:** Serviço distribuído e altamente escalável de armazenamento de objetos.
- **Adição do Great Expectations:** Serviço para análise da qualidade de dados.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

TDP 2.2 | 2024-S1

- **Adição do Kafka-UI:** Interface gráfica para monitoramento e gerenciamento de clusters Kafka.
- **Suporte estendido para sistemas operacionais:** CentOS/RHEL 8+ e 9+, Rocky Linux 8+ e 9+, e AlmaLinux 8+ e 9+.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

TDP 2.1 | 2023-S2

- **Adição do Apache Iceberg:** Formato avançado para tabelas analíticas que requerem alto desempenho com grandes volumes de dados.
- **Adição do Apache Flink:** Serviço de processamento de stream distribuído.
- **Adição do Apache Ranger KMS:** Serviço de gerenciamento de chaves de criptografia.
- **Atualizações de componentes:** Versões mais recentes de todos os componentes.

© Copyright Tecnisys 2026. All rights reserved.

TDP 2.0 | 2023-S1

- **Adição do Apache Airflow:** Serviço de orquestração de pipelines.
- **Adição do Apache Druid:** Serviço OLAP distribuído.
- **Adição do Apache Superset:** Serviço de exploração e visualização de dados.
- **Adição do Delta Lake:** Formato de tabela com suporte transacional completo.
- **Atualizações de componentes:** Versões mais recentes de todos os componentes.

TDP 1.0 | 2022-S2

- **Plataforma de Big Data baseada no Ecossistema Hadoop:** Com instalação, configuração e monitoramento integrados e centralizados
- **Repositório de pacotes exclusivo e dependências diretas:** Repositório Público de Pacotes da Tecnisys.
- **Compatibilidade para sistemas operacionais**: CentOS/RHEL 7.5

Documentação v3.0

Footnotes

1. Exclusivo para a edição TDP Kubernetes. ↩

© Copyright Tecnisys 2026. All rights reserved.

PARTE VIII - OUTRAS INFORMAÇÕES

© Copyright Tecnisys 2026. All rights reserved.

Suporte

A seguir, respondemos às dúvidas mais comuns sobre o suporte ao TDP realizado pela Tecnisys e seus parceiros.

Como obter suporte para o TDP?

A Tecnisys oferece suporte básico e gratuito a todos os usuários cadastrados.

Faça seu cadastro agora mesmo no site da Tecnisys para acessar a nossa página de suporte.

Se o seu negócio exige atendimento prioritário, garantias (SLA), disponibilidade técnica (24x7), atendimento inicial em até 15 minutos, monitoração avançado, análise de riscos e prevenção de incidentes, consultoria, entre outros serviços especializados, entre em contacto conosco.

Como relatar uma falha no TDP?

Na área de Suporte, você poderá abrir um chamado para informar a falha e acompanhar sua resolução.

O que é o Tecnisys Support Agent (TSA)?

O Tecnisys Support Agent é um serviço instalado junto com o Apache Ambari, para facilitar a identificação de máquinas e ambientes de dados subscritos no momento da abertura de chamados na Área de Suporte.

O serviço do TSA é executado em segundo plano e não interfere no desempenho do sistema operacional ou do banco de dados. Além disso, tal serviço não requer uma conexão direta com a Internet.

O arquivo de identificação da máquina (*subscription.info*), gerado diariamente pelo TSA, é mantido de forma segura e criptografada no diretório */opt/tecnisys-support-agent*.

Como instalar separadamente o Tecnisys Support Agent (TSA)?

Download do pacote

© Copyright Tecnisys 2026. All rights reserved.

Instruções

Para instalar separadamente o Tecnisys Support Agent, siga os passos abaixo:

1. Faça o seu cadastro gratuito no site da Tecnisys;
2. Acesse o Repositório Público de Pacotes da Tecnisys;
3. Clique no botão *Sign out*, localizado no canto superior direito da página;
4. Informe suas credenciais de acesso (as mesmas do site da Tecnisys);
5. Clique na opção *Browse*, localizada no menu lateral esquerdo;
6. Na área de navegação central, acesse: *yum >> tecnisys >> tsa*
7. Selecione o pacote de instalação (`.rpm`) da última versão disponível para o seu sistema operacional;
8. Clique no link da propriedade *Path*, localizada em *Summary*, na lateral direita da página, para iniciar o download do pacote.

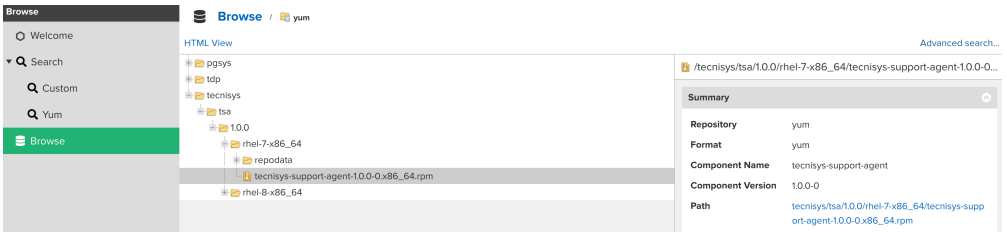


Figura 1 - Download do pacote de instalação do TSA

Instalação do pacote

Instruções

1. Instale o pacote de instalação do TSA na máquina subscrita, conforme o exemplo abaixo:

Terminal input

```
rpm -ivh tecnisys-support-agent-1.0.0-0.x86_64.rpm
```

© Copyright Tecnisys 2026. All rights reserved.

2. Após a instalação, execute o comando abaixo para ativar e iniciar o serviço do TSA:

Terminal input

```
systemctl enable --now tecnisys-support-agent.service
```

3. Para verificar o status do serviço do TSA, execute o comando abaixo:

Terminal input

```
systemctl status tecnisys-support-agent.service
```

Também é possível baixar o pacote de instalação do TSA diretamente do terminal, via curl ou wget. Veja o exemplo abaixo:

Instruções

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tecnisys/tsa/1.0.0/rhel-8-x86_64/tecnisys-
support-agent-1.0.0-0.x86_64.rpm
```

© Copyright Tecnisys 2026. All rights reserved.

Informações Legais

Esta documentação é mantida pela Tecnisys Informática e Assessoria Empresarial LTDA.

O TDP (Tecnisys Data Platform) é marca registrada da Tecnisys.

A Tecnisys não fornece representações para seus produtos.

O TDP (Tecnisys Data Platform) incorpora componentes de software de várias comunidades, em especial aquelas sob os termos de licenciamento da Apache Software License. Outros componentes podem ser incorporados sob termos alternativos. Estas informações podem ser obtidas diretamente nos arquivos de licença disponíveis nos sites de cada comunidade.

Para detalhes dos componentes e serviços que compõem o portfólio de serviços da Tecnisys, visite nossa página de suporte e vendas ou entre em contato.

O uso de softwares e componentes aqui descritos sujeitam-se às permissões e limitações definidas pela Apache Software License.

As informações aqui contidas estão sujeitas a alterações sem aviso prévio. A Tecnisys reserva-se o direito de incluir, excluir ou alterar qualquer componente aqui descrito, a qualquer tempo e sem aviso prévio.

A Tecnisys isenta-se de qualquer responsabilidade, garantia ou condição de qualquer tipo, expressa ou implícita, decorrente do uso dos produtos aqui descritos, salvo o que é definido por lei ou expressamente definido(s) em um contrato bilateral específico, por escrito. Nada aqui deve ser interpretado como constituindo uma garantia adicional.

A Tecnisys não garante ausência de interrupções, defeitos, erros, danos técnicos, erros editoriais, proteção contra perda, corrupção ou indisponibilidade de dados, omissões, exceto quando expressamente definido em contrato bilateral específico e por escrito.

As informações de direitos autorais podem ser encontradas na documentação que acompanha cada componente e sua versão específica.

Contato

© Copyright Tecnisys 2026. All rights reserved.

Site: www.tecnisys.com.br

E-mail: faleconosco@tecnisys.com.br

Telefone: +55 61 3039-9700

Endereço: Trecho 8, Lote 245, SIA, Brasília, DF, Brasil, CEP 71205-080

© Copyright Tecnisys 2026. All rights reserved.