TDP Documentação 2.2

Version (pt-PT)

TECNISYS

Índice

PARTE VII - ROADMAP

PARTE VIII - OUTRAS INFORMAÇÕES

PARTE I - INTRODUÇÃO

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

Com o TDP, é possível avançar da ingestão à análise de dados, criando pipelines integrados e fluxos de dados contínuos para diferentes tipos de *workloads*, seja *streaming* ou *em lote*. A Plataforma permite explorar e analisar rapidamente grandes volumes de dados, treinar modelos de aprendizagem de máquina, apresentar grafos 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 descarregar gratuitamente essa incrível Plataforma de Dados.

Por que usar o TDP?

O TDP possibilita a implementação dum 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..

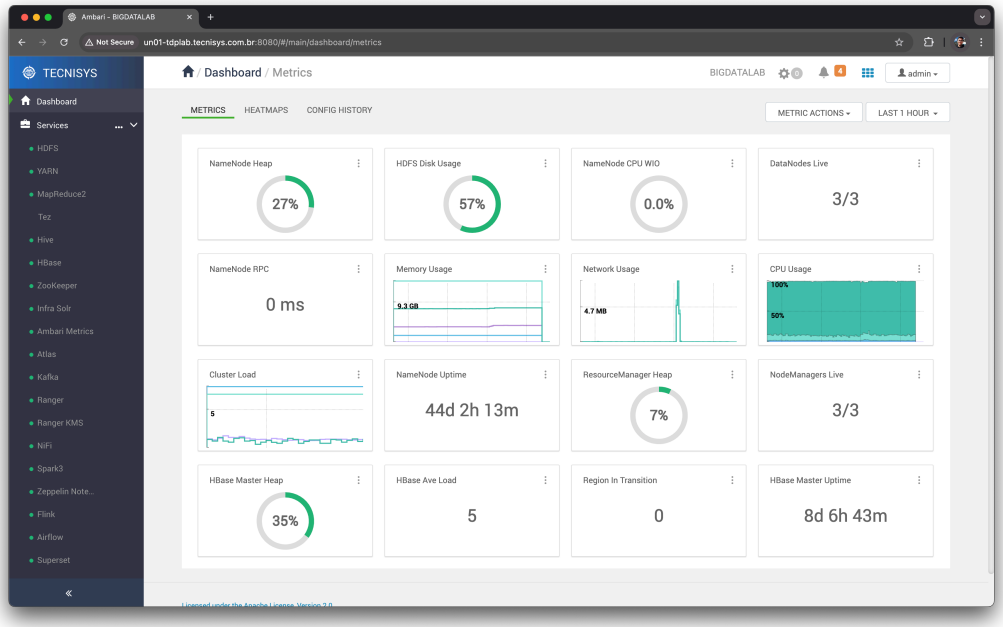


Figura 2 - Interface gráfica para a administração de máquinas e serviços

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

- 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, engenheiros de dados e especialistas atentos às necessidades organizacionais.

CONCEITOS

Apache Airflow

Orquestração de Pipelines

Pipeline de dados consiste num conjunto de tarefas ou ações que precisam ser executadas numa 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 relações entre operadores de forma programática.

Embora muitos Orquestradores de Pipelines 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.

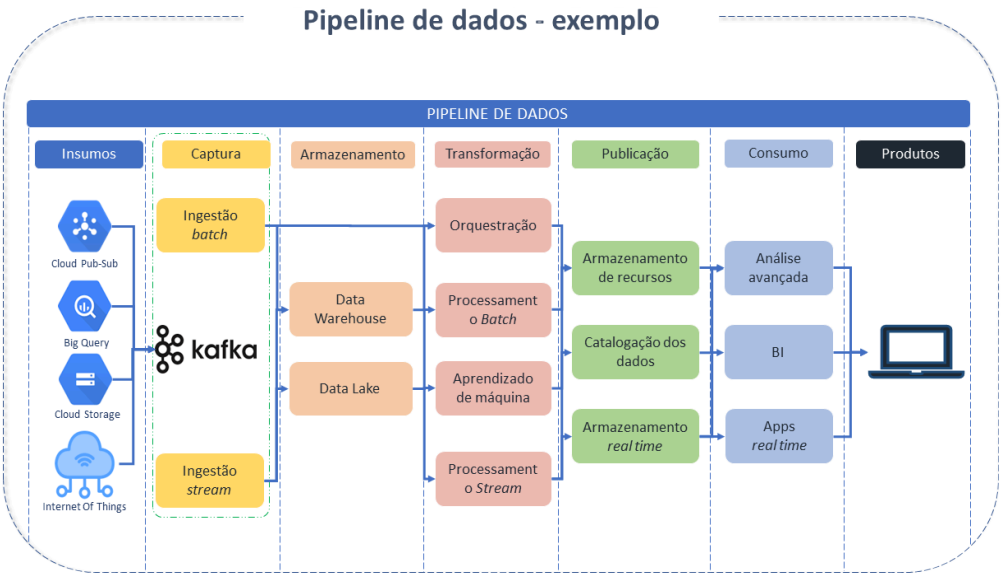


Figura 1 - Pipeline de dados

Características do Apache Airflow:

O Apache Airflow foi criado no Airbnb em 2014 como uma solução para gerenciar os 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 extensibilidade e adição de integrações com muitos sistemas diferentes. A comunidade Airflow já desenvolveu uma rica coleção de extensões para diferentes bases 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.
- **Interface Web Rica**: A rica interface web do Airflow disponibiliza visualização fácil para monitorização de resultados de execuções de *pipeline* bem como a depuração de quaisquer falhas.

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

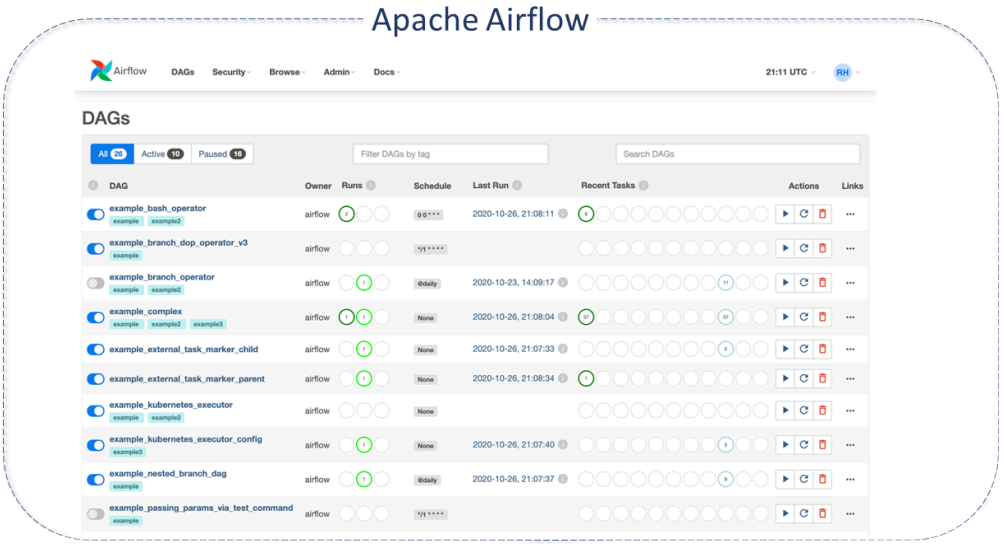


Figura 2 - Interface Apache Airflow

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).

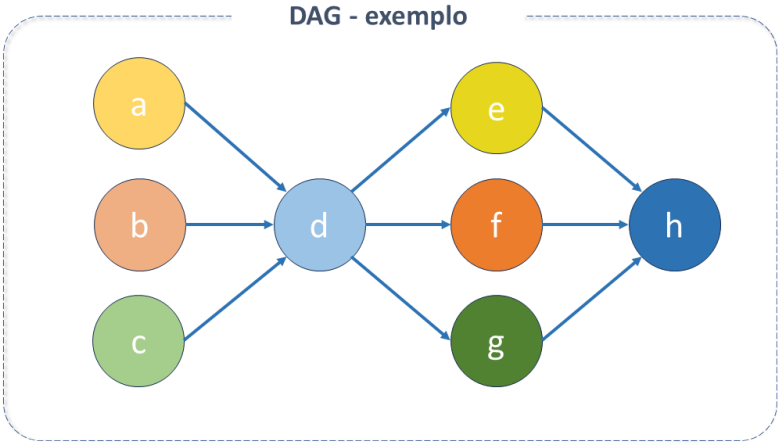


Figura 3 - Exemplo da estrutura dum DAG

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 ficheiros *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 num cronograma definido, como parte do *DAG*.

Seus parâmetros mais importantes são:

  - **dag_id**: identificador dum *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:

    - **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 num *Cluster Dask Distributed*. *Clusters Dask* podem executar numa 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 agendadas.
- **Ficheiros *DAG***: Armazenam os ficheiros que são lidos pelo Scheduler e Executor (e pelos *Workers* que o *Executor* tenha).
- **Base de Dados de metadados**: Bases de dados SQL usado pelo Scheduler, Executor e Webserver para armazenar metadados sobre os *pipelines* de dados que estão a ser executados.

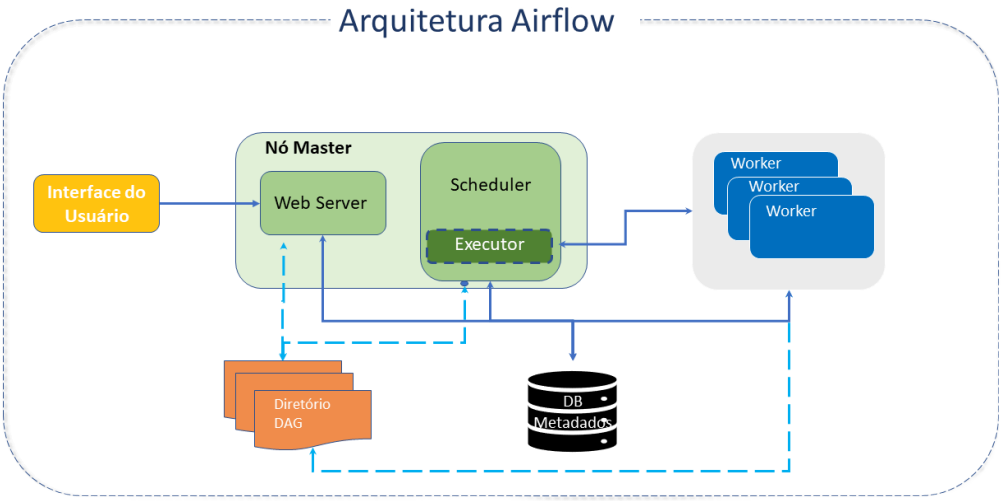


Figura 4 - Arquitetura Airflow

Necessidade de um Diretório Partilhado para DAGs no Airflow

Num ambiente distribuído do Apache Airflow, especialmente quando é utilizado o Celery Executor, Kubernetes Executor ou Dask Executor, é essencial garantir que todos os Workers tenham acesso consistente aos mesmos ficheiros de DAG.

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

Quando existem vários Workers a executar tarefas em paralelo, todos eles precisam de conseguir ler o mesmo conjunto de DAGs.

Por defeito, o Scheduler e o Webserver lêem os ficheiros diretamente a partir do diretório: `/usr/tdp/current/airflow/dags`.

No entanto, se este diretório não for partilhado entre os nós, cada Worker verá apenas as DAGs locais — resultando em falhas de execução ou na ausência das DAGs na interface Web.

Diretório Partilhado 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 (normalmente onde o Airflow Scheduler está instalado) e montando-o nos restantes hosts que executam o Airflow Worker.

Desta forma, todos os componentes do Airflow — Webserver, Scheduler e Workers — acedem exatamente ao mesmo repositório de DAGs.

Exemplo de Configuração

1. Instalar o servidor NFS e exportar o diretório:

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

   Terminal input

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

   1.2 Criar o diretório de DAGs caso não exista:

   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. Editar o ficheiro `/etc/exports` e adicionar a linha abaixo (ajustar a rede conforme o ambiente):

   2.1 Edição do ficheiro:

   Terminal input

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

   2.2 Exportar e ativar o serviço:

Terminal input

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

2.3 Verificar se a partilha está ativa:

Terminal input

```
sudo exportfs -v
```

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

3.1 Instalar o cliente NFS e montar o diretório partilhado:

Terminal input

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

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

Terminal input

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

3.3 Montar 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 reinício, adicionar a entrada no `/etc/fstab`:

Terminal input

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

Recursos do Apache Airflow

- **Pooling**: Os pools são uma funcionalidade adicional que disponibiliza mecanismos de gerenciamento de recursos. Os pools limitam a execução paralela 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 num ambiente ou recurso específico.

- **Plugins**: O Airflow tem um gerenciador de *plugins* simples que pode integrar recursos externos ao seu núcleo simplesmente "soltando" ficheiros 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 gerem o estado de operação dos *operators* (definem uma unidade de trabalho dentro dum *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

Workflow e geralmente (não sempre) é atómico, 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" dum 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.

> **NOTA**
>
> - Se dois operadores precisarem partilhar informações, podem ser combinados num 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 dependências 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.
- ***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 a sua última execução.

Boas Práticas para o Apache Airflow

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

- **Escrever um *DAG***: Embora a criação dum 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 ficheiros alterados. É recomendável que se evite acionar DAGs logo após a sua alteração ou a alteração de ficheiros 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, a base 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*. Os seus fluxos de trabalho são criados por meio de *scripts Python*. Foi projetado sob o princípio de "configuration as code". Embora existam outras ferramentas que adotem este princípio, o uso do *Python* é o que permite aos desenvolvedores a importação de bibliotecas e classes.

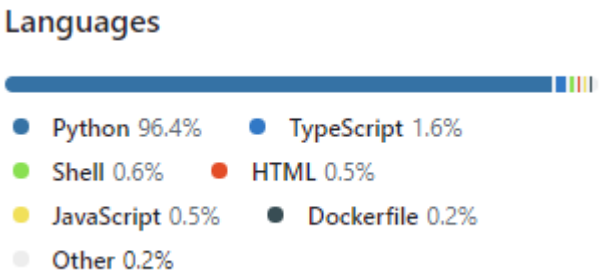


Figura 5 - Linguagens do Airflow

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão actual 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).

Apache Ambari

Gestão do Cluster

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

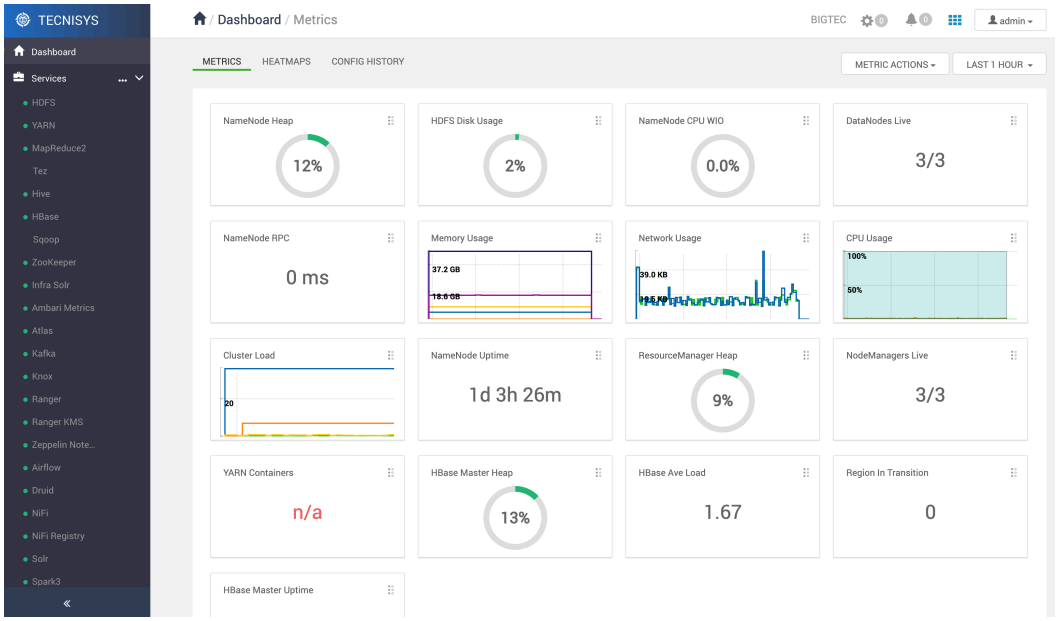


Figura 1 - Exemplo do Ambari

A sua arquitetura consistente, API REST robusta e interface web intuitiva e interativa disponibilizam 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.

Objetivos

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

- **Plataforma Independente:** O sistema deve suportar arquitetonicamente qualquer hardware e sistema operativo. Componentes que são inerentemente dependentes de uma plataforma devem ser conectáveis com interfaces bem definidas.
- **Componentes Conectáveis:** 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.
- **Gestão de Versões e Atualizações:** 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 permitir 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 Falhas:** O sistema deve ser capaz de se recuperar de qualquer falha nos 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 num estado consistente.
- **Segurança:** A segurança implica 1) autenticação e autorização baseada em perfis de usuários (API e interface web), 2) instalação, gerenciamento e monitorização da Plataforma de Serviços protegida via Kerberos e 3) garantia de autenticação e encriptação da comunicação na rede entre os componentes do Ambari.
- **Rastreamento de Erro:** A arquitetura deve esforçar-se 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 Intermédio e Operações em *Near Real Time*:** Para operações que demoram um pouco para serem concluídas, o sistema precisa ser capaz de disponibilizar feedback ao usuário com progresso intermédio em relação à execução atual de tarefas, percentagem de operação concluída e referência ao log de operação, em tempo hábil (quase em tempo real).

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 dum 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 dum 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 dum DataNode no nó N1 e a instalação dum 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.

- Ação (Action): Uma "ação" consiste numa tarefa ou tarefas numa 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 dum 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 numa 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 registo.

- 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 controlo do *Cluster*. Consiste de vários *entry-points* disponíveis para diferentes necessidades, tais como: - Gestão 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 configuração preliminar do Ambari.
  - Configuração de Autenticação e Autorização: entry-point para configuração do mecanismo de gerenciamento de identidades, autenticação e autorização. Opções

disponíveis: LDAP (Lightweight Directory Access Protocol), PAM (Pluggable Authentication Module) e Kerberos.
-*Backup* e_Restore_: entry-point para a realização de cópia de segurança (snapshot) e_Restore_ da instalação atual, com exceção da base de dados.

- **Ambari Agent:** Componente Slave responsável pela execução de ações e envio de informações e métricas de determinado host.
- **Base de Dados:** Área de persistência do estado, métricas e metadados da infraestrutura do *Cluster* (serviços e hosts). O Ambari oferece suporte a vários Sistemas Gestores de Base de Dados Relacional (SGBDR), tais como PostgreSQL (default), MySQL, MariaDB, Oracle e Microsoft SQL Server, cuja escolha é feita durante a primeira configuração do Ambari Server.

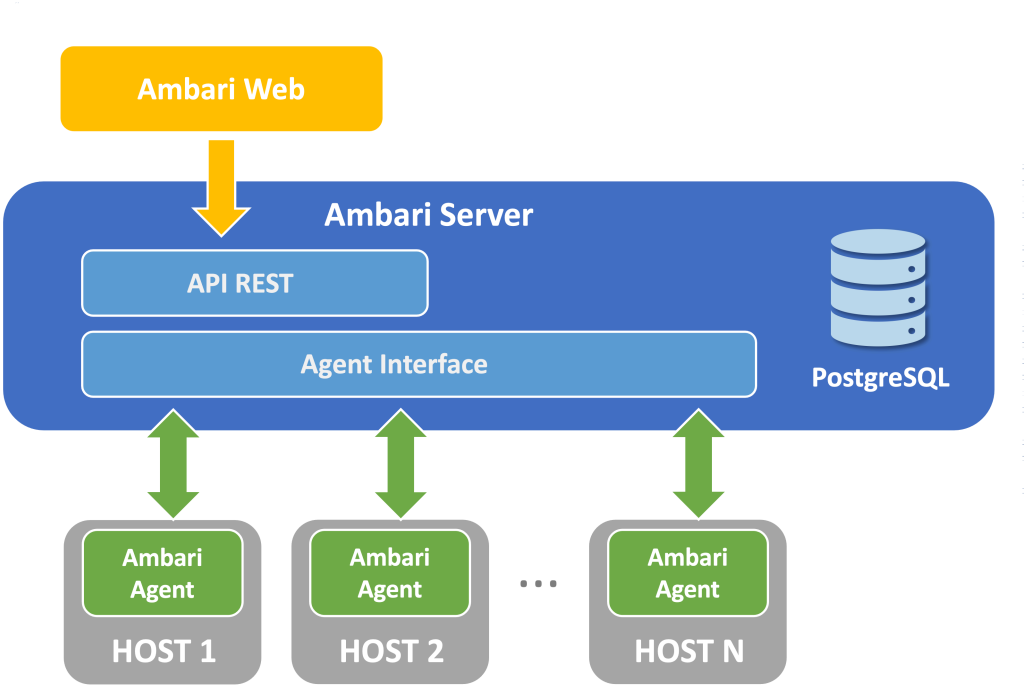


Figura 2 - Arquitetura do Ambari

Detalhes sobre o projeto

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

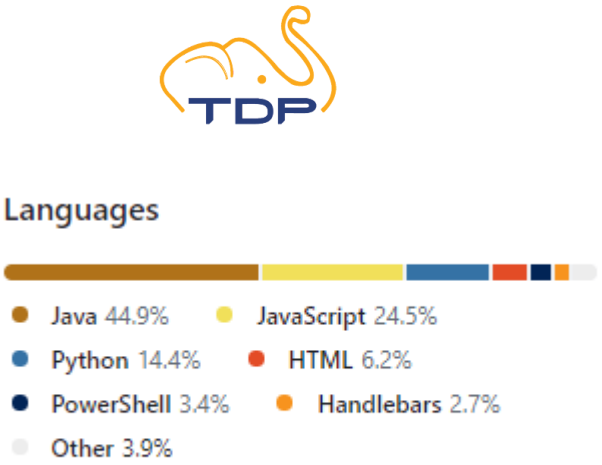


Figura 3 - Linguagens predominantes do projeto Apache Ambari

Fontes:

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

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 utilizados 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 para as transformações digitais e otimize os parques de tecnologia, demanda das organizações um "alicerce" estruturado e grande 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 regulatória, na redução de custos com políticas e sistemas centralizados, no crescimento dos dados controlados e organizados, protegendo-os de qualquer contratempo, dentro dum ambiente cultural favorável à inovação e permeando todo o *pipeline* de tecnologias pelas quais os dados circularão.

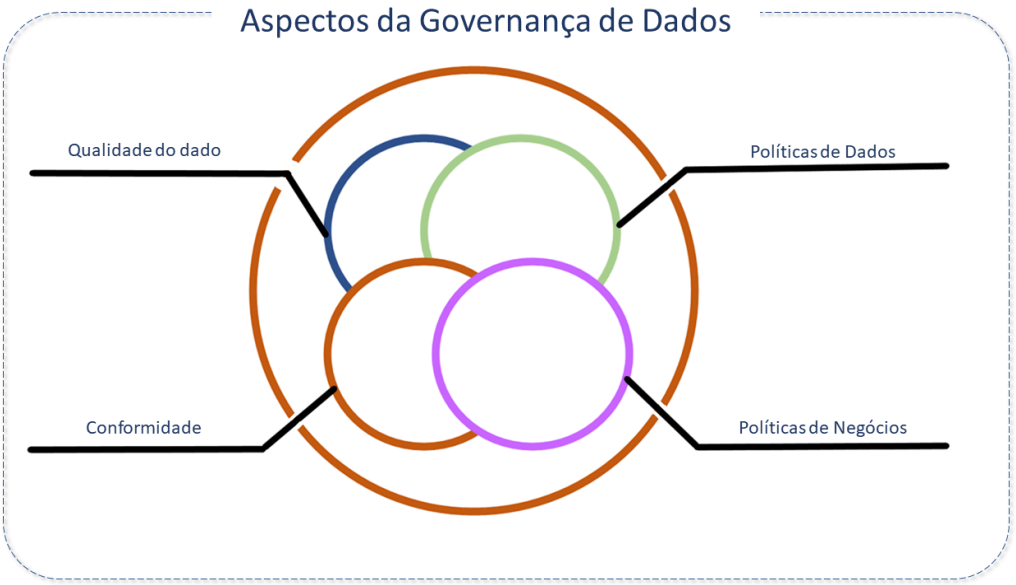


Figura 1 - Aspectos da Governança de dados

Características do Apache Atlas

O Apache Atlas é um conjunto escalável de serviços essenciais de governança fornecidos para o Hadoop, com o objetivo de auxiliar as organizações a cumprir os 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 controlos de governança independentes de plataforma atendendo eficazmente 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.
- **Controlo de acesso a dados orientado por metadados.**
- **Modelagem flexível de dados comerciais e operativos.**
- **Classificação de dados:** auxiliando no entendimento da natureza dos dados e sua classificação com base em fontes externas e internas.

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

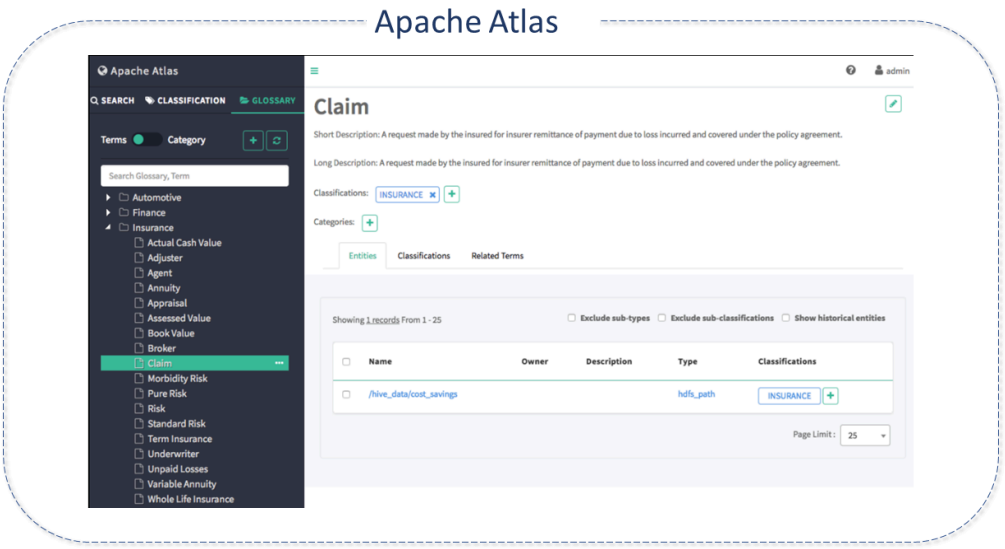


Figura 2 - Interface Apache Atlas

Arquitetura do Apache Atlas

A arquitetura do Apache Atlas envolve os seguintes componentes:

Core

- Type System: Permite a definição dum 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 das relações entre os objetos de metadados.

    - A engine *Graph* é responsável pela tradução entre *types* e *entidades* do *Type System* e pelo modelo de persistência do grafo 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 (´tempo real´ ou ´em lote´).

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.

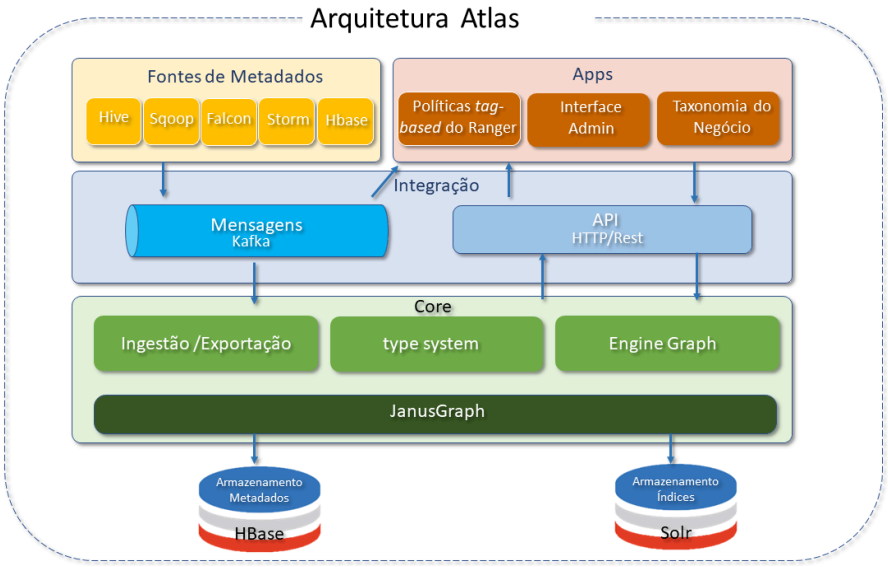


Figura 3 - Arquitetura Apache Atlas

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 operativos 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 aceder e atualizar a linhagem.
- **Integração com Ranger:**: Integração com Apache Ranger habilitando autorização/mascaramento de dados no acesso ao dado, baseado em classificações

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 dum 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, num ambiente não assegurado, esta identidade é a mesma que o usuário autenticado no S.O. para iniciar o servidor. No entanto, num *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 dum nome de usuário fornecido.

    - 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**::

    - **Controlos 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

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

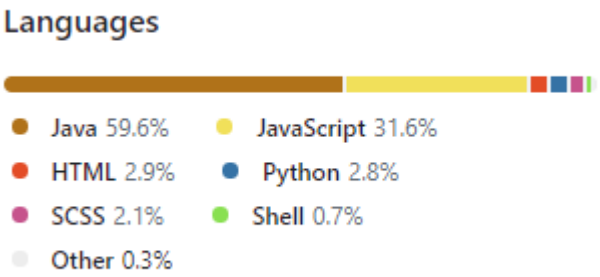


Figura 4 - Linguagens do Atlas

Fontes:

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

Delta Lake

Estrutura de Armazenamento

Nas arquiteturas modernas de dados, a **camada de armazenamento** é a base que permite guardar grandes volumes de dados de forma escalável e económica. Utiliza sistemas como Amazon S3, Azure Data Lake Storage ou HDFS para armazenar ficheiros em formatos como Parquet, ORC ou Avro. Contudo, esta camada não impõe estrutura, controlo de versão, validação de esquema ou transações — apenas armazena dados.

Sobre esta base, surgiu o conceito de **Data Lake**: uma arquitetura concebida para armazenar dados brutos em larga escala, provenientes de múltiplas fontes e em diferentes formatos. A promessa dos data lakes foi centralizar tudo num único local — facilitando análises futuras, integrando dados variados e reduzindo custos. Tornaram-se populares em projetos de ciência de dados e pipelines de ingestão massiva.

Porque os Data Lakes se Tornaram Populares

Mesmo com limitações técnicas, os data lakes tornaram-se populares pela sua flexibilidade e economia. Permitem:

- **Armazenar os dados no seu formato original**, sem necessidade de modelação prévia.
- **Capturar dados em batch e em tempo real**, com escalabilidade praticamente ilimitada.
- **Servir múltiplos perfis e ferramentas de utilizador**, suportando análise ad hoc, machine learning, relatórios e dashboards.
- **Consolidar e democratizar o acesso a dados**, quebrando silos de sistemas legados.

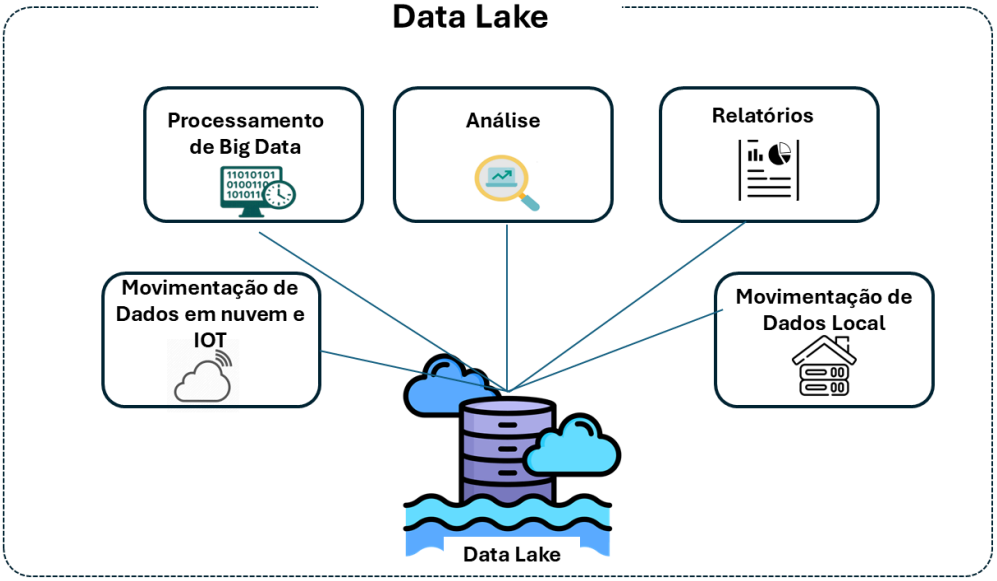


Figura 1 - Data Lake

Limitações dos Data Lakes

Apesar das vantagens, os data lakes tradicionais não oferecem os controlos necessários para ambientes analíticos fiáveis:

- **Sem transações ACID**: múltiplas escritas podem gerar inconsistência ou corrupção.
- **Sem controlo de versões dos dados**: não é possível consultar ou restaurar um estado anterior da tabela.
- **Sem validação de consistência de esquemas**: erros estruturais podem passar despercebidos, comprometendo as análises.
- **Sem otimizações de desempenho para consultas**: ficheiros pequenos e desorganizados afetam o tempo de resposta.

Estas limitações impedem os data lakes de serem uma base confiável para aplicações analíticas críticas. Para ultrapassar estas deficiências, foi criada uma nova camada — o **Delta Lake**.

O que é o Delta Lake

O **Delta Lake** é uma camada transacional que atua sobre um data lake existente. Não substitui o data lake — complementa-o. Ao adicionar transações ACID, controlo de versões, gestão de esquemas e integração com ferramentas como Apache Spark, o Delta Lake transforma os data lakes brutos numa plataforma mais robusta, fiável e auditável — conhecida como **Lakehouse**[1].

Funcionalidades do Delta Lake

O Delta Lake introduz funcionalidades essenciais que resolvem as falhas dos data lakes tradicionais e permitem uma arquitetura de dados escalável e fiável. Entre os principais destaques estão:

- **Transações ACID**: garantem atomicidade, consistência, isolamento e durabilidade mesmo em ambientes distribuídos. Leituras e escritas são seguras e previsíveis.
- **Controlo de Versões e Time Travel**: todas as alterações são automaticamente versionadas, permitindo consultas a estados anteriores, rollback e auditorias históricas.
- **Gestão e Validação de Esquemas**: evita a ingestão de dados com estruturas incorretas. O esquema pode evoluir de forma controlada sem comprometer a integridade da tabela.
- **Unificação de Batch e Streaming**: a mesma tabela Delta pode receber dados batch e streaming, garantindo consistência entre os modos de ingestão.
- **Desempenho Otimizado para Consultas**: técnicas como compactação de ficheiros pequenos, ordenação por colunas (Z-order) e *data skipping* aceleram significativamente as consultas.
- **Integração com o Ecossistema Spark**: permite operações SQL, APIs estruturadas e processamento em tempo real diretamente sobre tabelas Delta.

Arquitetura do Delta Lake

A arquitetura do Delta Lake apoia-se no data lake tradicional, introduzindo uma camada transacional que garante fiabilidade e desempenho. Esta camada é composta por três elementos principais:

- **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. Esta camada física continua a oferecer escalabilidade e economia.
- **Delta Log**: um log de transações que regista todas as alterações efetuadas nas tabelas. Este log possibilita o controlo de versões, garantindo que cada modificação seja rastreada e permitindo funcionalidades de time travel e auditoria.

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

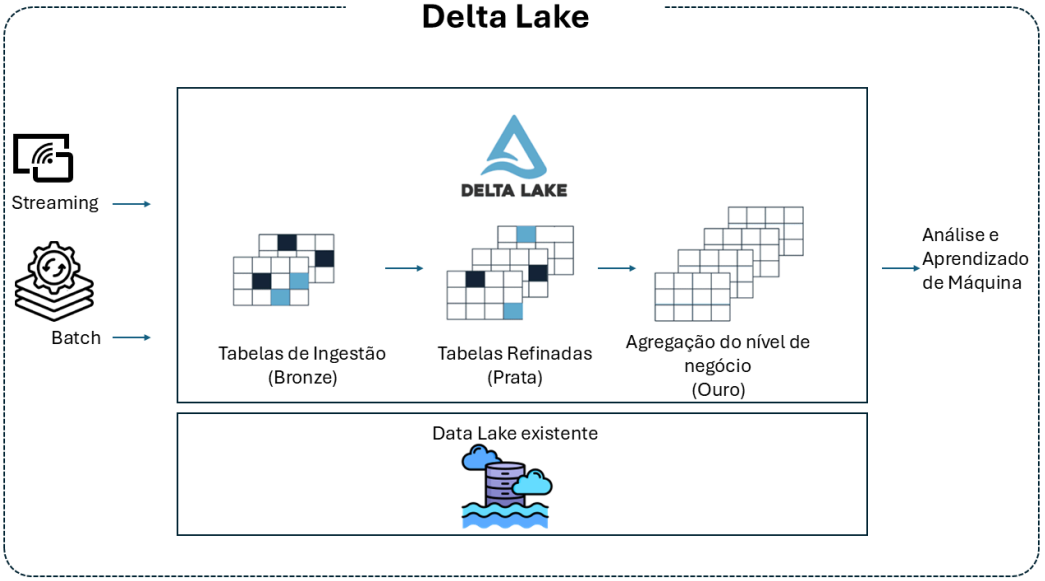


Figura 2 - Arquitetura do Delta Lake

Como Funciona o Delta Lake

O Delta Lake atua continuamente sobre o armazenamento físico, gerindo alterações de dados e garantindo transações seguras e rastreáveis. O seu funcionamento organiza-se em três áreas principais:

Ingestão de Dados

O Delta Lake suporta múltiplos modos de ingestão:

- **Streaming**: integra-se com o Apache Spark Structured Streaming para ingestão contínua, permitindo atualizações em tempo real com semântica exactly-once.
- **Batch**: suporta ingestão em grande escala com operações como `MERGE`, `UPDATE`, `DELETE` e `INSERT`, sem comprometer a integridade das tabelas.

Processamento de Consultas

As tabelas Delta podem ser consultadas através de:

- **SQL**: para consultas interativas ou analíticas diretamente sobre dados versionados.

- **APIs de alto nível**: compatíveis com as APIs do Apache Spark (DataFrame, SQL, Structured Streaming).
- **Time Travel**: é possível navegar no Delta Log e consultar o estado dos dados em qualquer ponto no tempo.

Gestão de Dados

O Delta Lake oferece mecanismos de otimização contínua:

- **Compactação de Ficheiros Pequenos**: operações como `OPTIMIZE` reorganizam os dados para melhorar o desempenho das consultas.
- **Ordenação Z-order**: organiza os dados por colunas de alta seletividade, reduzindo custos de leitura.
- **Limpeza de Dados Obsoletos**: com `VACUUM`, ficheiros não referenciados são removidos com segurança, libertando espaço e mantendo o desempenho.

Estas funcionalidades trabalham de forma integrada, mantendo a integridade dos dados mesmo sob alta concorrência ou ingestão contínua.

Capacidades do Delta Lake

- **Transações ACID com Spark**: níveis de isolamento serializáveis garantem que os leitores nunca vejam dados inconsistentes.
- **Gestão Escalável de Metadados**: utiliza o processamento distribuído do Spark para gerir metadados de tabelas com petabytes de dados e biliões de ficheiros.
- **Unificação de Streaming e Batch**: uma tabela Delta é simultaneamente uma tabela batch e uma fonte/sumidouro de streaming. Ingestão contínua, backfill batch e consultas interativas funcionam de forma integrada.
- **Validação de Esquema**: lida automaticamente com variações de esquema para evitar inserção de registos inválidos durante a ingestão.
- **Time Travel**: versionamento de dados permite rollback, auditoria histórica completa e experiências de machine learning reproduzíveis.
- **Upserts e Deletes**: suporta operações de `merge`, `update` e `delete`, ideais para cenários como change data capture, slowly changing dimensions (SCD) e upserts em streaming.
- **Ecossistema de Conectores**: estão disponíveis conectores para ler e escrever em tabelas Delta a partir de engines como Apache Spark, Flink, Hive, Trino, AWS Athena, entre outros.

Quando Utilizar Delta Lake

O Delta Lake é indicado para cenários que requerem:

- **Dados em vários formato**s provenientes de diversas fontes;
- **Utilização dos dados em várias tarefas posteriores distintas**, como análise, ciência de dados, aprendizagem automática, etc.;
- **Flexibilidade** para executar diversos tipos de consultas sem a necessidade de formular as perguntas antecipadamente;
- **Processamento de Dados em Tempo Real**: ingestão e análise em tempo real com garantias de consistência.
- **Gestão de Grandes Volumes de Dados**: suporta petabytes de dados com desempenho otimizado.
- **Necessidade de Auditoria e Versionamento**: acesso a versões anteriores dos dados para auditoria e recuperação.

Quando Delta Lake Pode Não Ser a Melhor Escolha

O Delta Lake pode não ser a melhor opção para:

- **Sistemas OLTP de Alta Concorrência**: aplicações que requerem transações de baixa latência e alta concorrência podem beneficiar mais de bases 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 concebido para integração completa com o Structured Streaming, permitindo operações unificadas sobre dados batch e streaming e suportando processamento incremental escalável. Permite:

- **Ingestão e Processamento em Tempo Real**: com Spark Structured Streaming.
- **Consultas SQL e Operações DML**: com suporte a transações ACID.
- **Otimizações de Desempenho**: tirando partido das capacidades de processamento distribuído do Spark.

Diferenças Entre Delta Lake e RDBMS Tradicionais

- **Armazenamento**: o Delta Lake usa ficheiros Parquet em sistemas de ficheiros distribuídos, enquanto os RDBMS usam blocos de armazenamento.

- **Transações**: o Delta Lake fornece transações ACID em ambientes distribuídos; os RDBMS em sistemas centralizados.
- **Evolução de Esquema**: o Delta Lake permite evoluções flexíveis, enquanto os RDBMS exigem alterações estruturais mais rígidas.

Boas Práticas com Delta Lake

- **Compactação de Ficheiros**: usar o comando `OPTIMIZE` para melhorar o desempenho.
- **Ordenação Z-order**: aplicar em colunas frequentemente filtradas.
- **Gestão de Versões**: usar time travel para auditorias e recuperação de dados.

Detalhes do Projeto Delta Lake

- **Linguagem de Programação**: desenvolvido em Scala e Java.
- **Licença**: código aberto sob licença Apache 2.0.
- **Integração**: compatível com Apache Spark, 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 actual é **4.0.0**, distribuída pelo Helm Chart `tdp-deltalake` v3.0.1.
>
> 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 numa única plataforma. Usa o armazenamento económico e escalável de um data lake, oferece transações ACID, versionamento, validação de esquema e catálogos como num data warehouse, suportando processamento batch e

streaming num mesmo local, além de permitir SQL e machine learning sobre os mesmos dados. ↩

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 num *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 uma Base de Dados com poder de processar grandes quantidades de dados em tempo real.

As bases de dados analíticas 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.

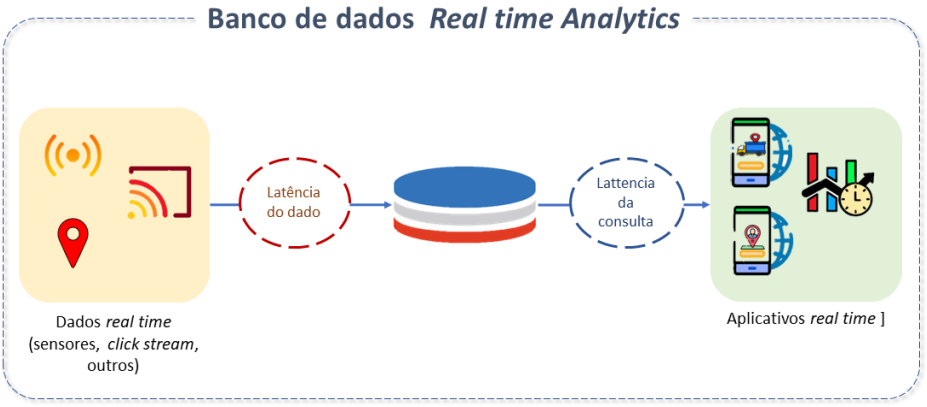


Figura 1 - Aspectos da Governança de dados

Características do Apache Druid

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

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 o time de dados de uma empresa de TI resolveu criar seu própria base de dados, após tentar a utilização 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 princípio orientador do Druid é efetuar operações apenas quando estritamente necessário:

    - 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 Base de Dados para GUIs ("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 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.

- **Alto Desempenho:** 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 a relação exclusiva 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 intermédios 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 monitorização 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).

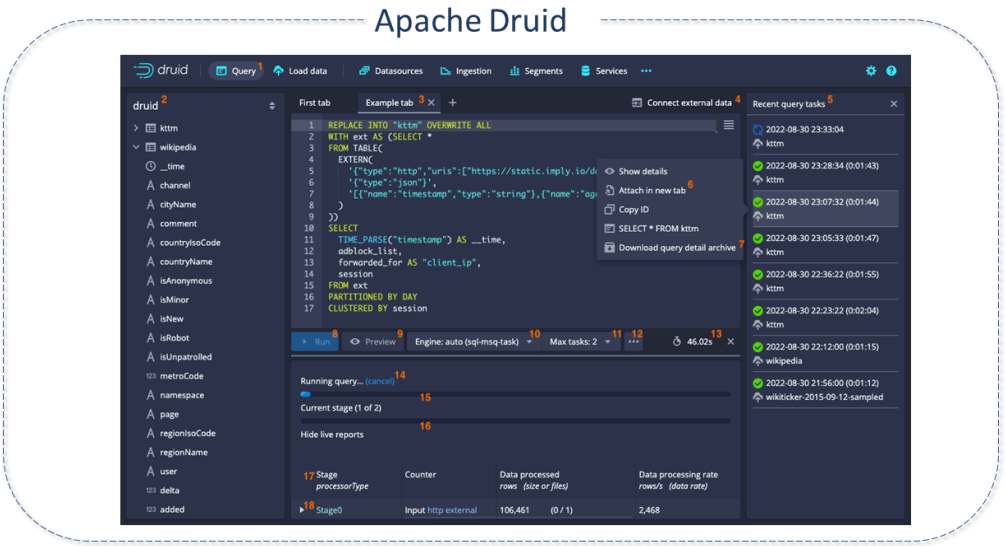


Figura 2 - Interface Apache Druid

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, Bases 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

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

> **NOTA**
>
> 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 partilhados, como informações de uso de segmentos e informações de tarefas. Numa implantação em *Cluster*, geralmente é um *RDBMS* tradicional, como PostgreSQL. Numa implantação de servidor único, pode ser uma Base de Dados Apache Derby local.
- *Zookeeper*: Usado para gerenciamento do *Cluster ativo*, coordenação e eleição de líder.

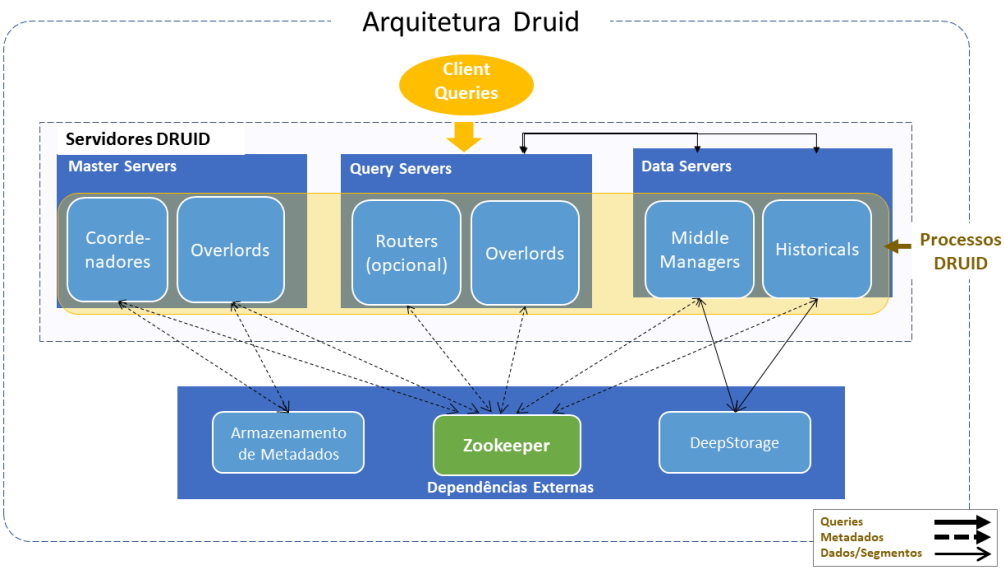


Figura 3 - Arquitetura Druid

Como o Apache Druid funciona

Design de Armazenamento

- ***Datasources* e Segmentos**:

  Os dados Druid são armazenados em *datasources*, que são similares a tabelas dum *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 dum *chunk*, dado é particionado num ou mais segmentos. Cada segmento é um ficheiro único. Uma vez que os segmentos são organizados em blocos de tempo, algumas vezes é adequado pensar segmentos *vivendo* numa 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 ficheiro de dados compacto e indexado:

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

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 um *timestamp* ISO8601 correspondente a quando o conjunto de segmentos foi iniciado). Serve para o controlo 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 ficheiros 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 num servidor de dados Druid, como uma tarefa em tempo real ou um processo histórico.

Ingestão:

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 ficheiros 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 a ser 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* e número de sequência para garantir a ingestão de uma só vez. O supervisor supervisionará o estado das tarefas de indexação para:

        - 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 dum *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 extensibilidade *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 `em lote`. 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

As operações de gerenciamento 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 *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.

- **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.

- 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, ficheiros simples ou armazenamento de objetos como o Amazon S3.

Quando o Apache Druid não se aplica

- Atualizações de baixa latência dos registos 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 dum 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 intermédios 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.

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 gerenciamento 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 Base 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 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 ficheiro 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.

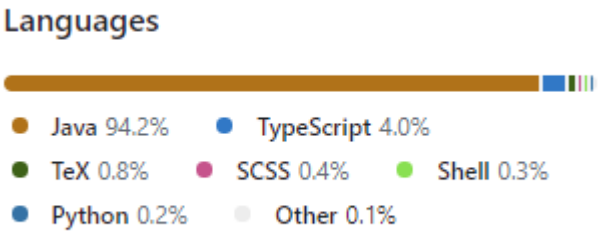


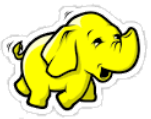
Figura 4 - Linguagens do Druid

Fontes

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

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.

> **NOTA**
>
> 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.

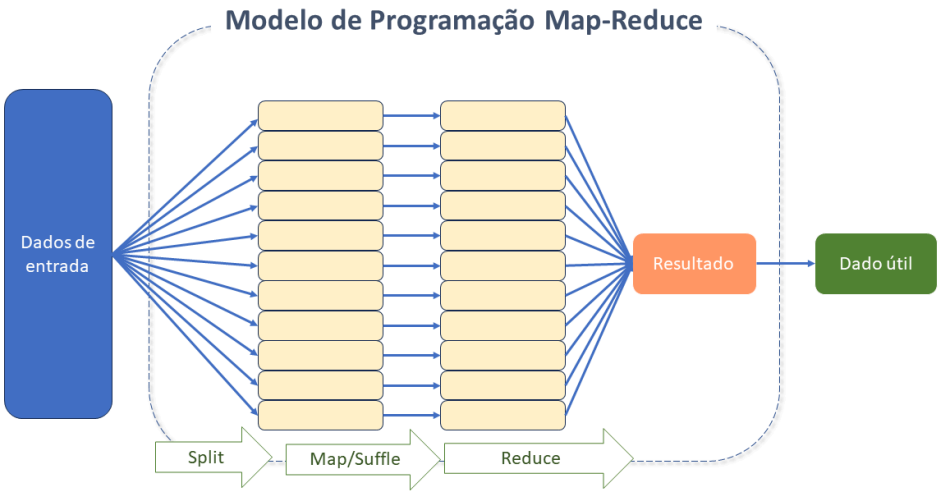


Figura 1 - Paradigma Map Reduce

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 ficheiros de configuração. O esforço para preparação dum 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 partilhando 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, disponibilizando, assim, alta disponibilidade num *Cluster* de computadores.

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 (ficheiros e *scripts* Java Archive (JAR)).
- **HDFS:** É um Sistema de ficheiros distribuídos, escalável, projetado para armazenar ficheiros muito grandes e disponibilizar 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 dum 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):

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

- 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 a 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.

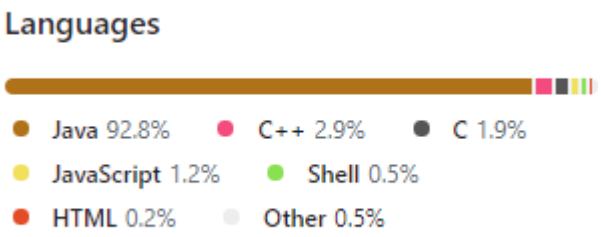


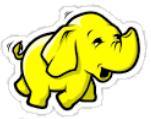
Figura 3 - Linguagens do Hadoop

Fonte(s):

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

Apache HDFS

Sistema de Ficheiros Distribuídos



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

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

O DFS agrupa vários nós de memória 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.

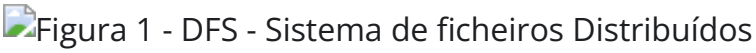


Figura 1 - DFS - Sistema de ficheiros Distribuídos

Características do Apache HDFS

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

Trata-se dum sistema de ficheiros 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 ficheiros 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 dum sistema de ficheiros, 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 ficheiro 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 disponibilizar elevada *largura de banda agregada* , escalar para centenas de nós num único *Cluster* e suportar dezenas de milhões de ficheiros numa única instância.

- *Modelo de Coerência Simples*:
  O HDFS foi projetado com base no princípio *write once read many*(escrita única, múltiplas leituras). Depois que os dados são gravados, não há suporte para atualizações num ponto arbitrário ou alterações exceto em *appends* e *truncates*(anexar o conteúdo ao final dos ficheiros 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.

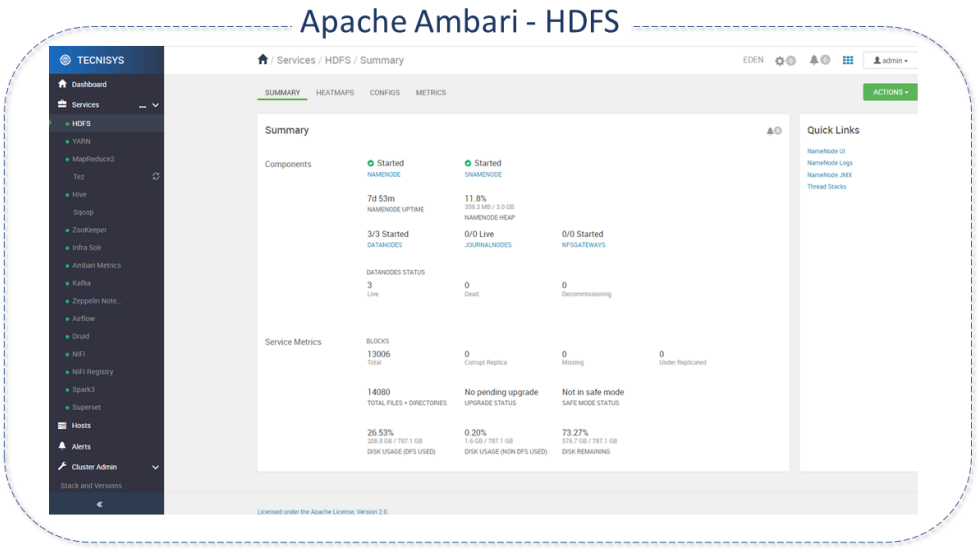


Figura 2 - Interface Ambari/HDFS

Arquitetura do Apache HDFS

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

> **NOTA**
>
> Quando o sistema é inicializado, o NameNode carrega toda informação do *fsimage* para sua memória, registando o estado atual do sistema. Depois, regista as modificações por meio do ficheiro *edit log* para atualizar o sistema.

- Os aplicativos *cliente* conversam com o "Namenode" sempre que desejam localizar ou trabalhar com um ficheiro(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 registada no ficheiro de *edit log* e, depois, o *filesystem namespace* é modificado(onde ficam as informações sobre os ficheiros). O *filesystem namespace* é armazenado localmente.

- O NameNode executa operações de namespace do sistema de ficheiros, como abrir, fechar, renomear ficheiros e diretórios. Além disso, qualquer alteração no namespace é registada pelo NameNode.

- Um *Cluster* HDFS consiste num 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 escrita dos clientes do sistema de ficheiro 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 o desempenho. +
- Como são considerados pontos de falha, periodicamente enviam relatórios de funcionamento para o Namenode.

> **NOTA**
>
> 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 memória e para leitura, evitando, assim, o direcionamento do cliente para DataNodes que não estão em funcionamento.

Os Namenodes e Datanodes são peças de *software* projetadas para executar em máquinas comuns. Estas máquinas geralmente executam um sistema operativo (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 numa implantação real este caso é muito raro.

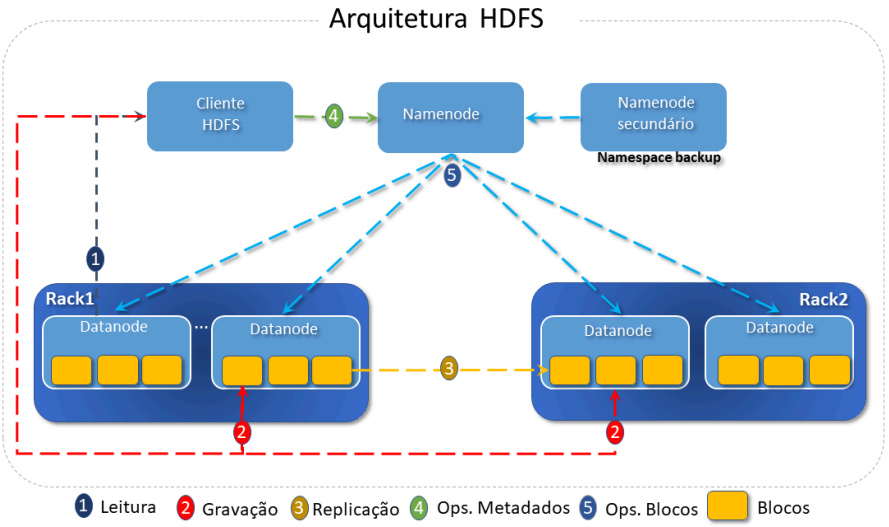


Figura 3 - Arquitetura HDFS

Recursos do Apache HDFS

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

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

- ***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 ficheiros e diretórios que partilha muito do modelo POSIX.

- **Data Replication(Replicação de Dados)**:
  O HDFS foi projetado para armazenar grandes ficheiros em máquinas dum grande *Cluster*. Cada ficheiro 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 ficheiro.
Todos os blocos dum ficheiro, 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 dum ficheiro. 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 num *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.

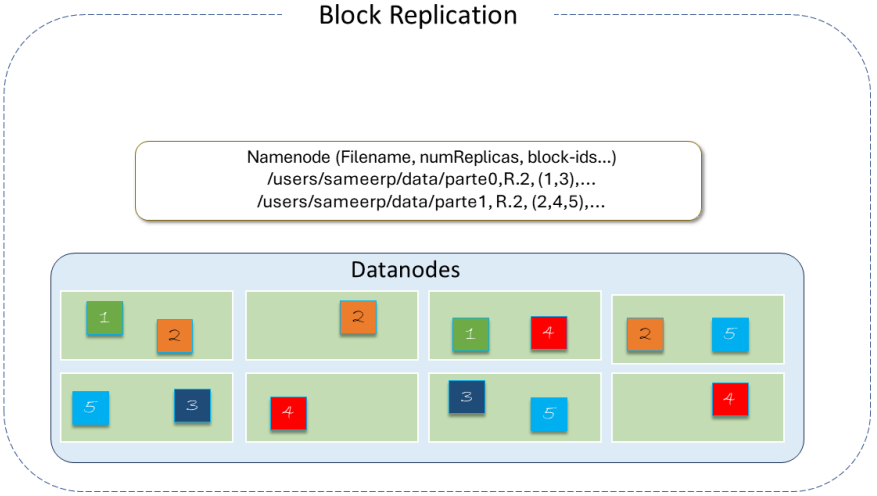


Figura 1 - Replicação de Blocos

- **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 ficheiros 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 *centro de dados* 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 num DataNode, caso contrário, num datanode aleatório no mesmo rack que o gravador, outra replica num nó num rack remoto diferente, e a última num 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 uma percentagem 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 ficheiros. 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 ficheiro de *log* de edições. Opera apenas em ficheiros e não exige que o *Cluster* Hadoop esteja em execução.

- **ImageViewer**: Ferramenta que "despeja" o conteúdo dos ficheiros *fsimage* do HDFS num formato "legível" e provê uma API *WebHDFS* somente leitura para analise e exame *offline* do NameSpace dum *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 ficheiros 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, desempenho, e tolerância a falhas.

- **Suporte à armazenamento de memória:** O HDFS suporta a escrita 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 num *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 transparente.

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

- **Observer NameNode:** Num *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 dum novo NameNode chamado "Observer NameNode". Da mesma forma que o NameNode "standby", o *Observer NameNode* se mantém atualizado em relação ao NameSpace e informações 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 num 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* dum ficheiro ou diretório) ou *soft links*(ou link simbólico - atuando como ponteiro ou referencia para o nome do ficheiro). 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 numa ampla variedade de máquinas. Qualquer máquina que suporta Java pode executar o software NameNode ou DataNode.

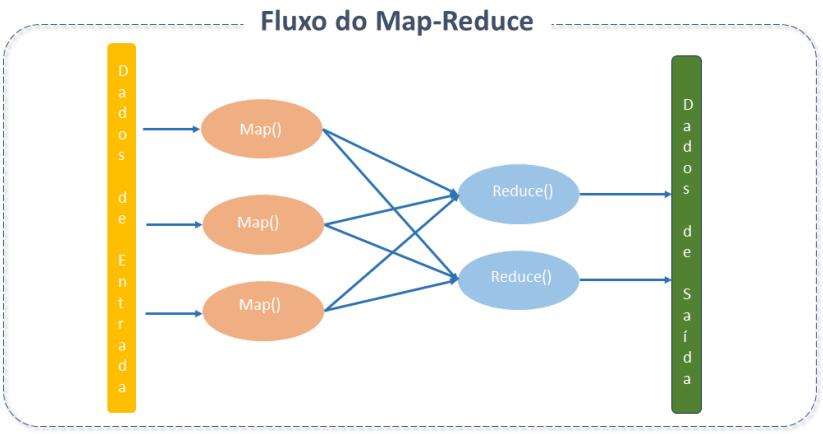
fonte(s): Hadoop Documentation

Apache 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 dum 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

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 num sistema de ficheiros. 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 disponibilizam 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 a monitorização, disponibilizando *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 ficheiro armazenado no *HDFS*, que pode ser de qualquer tipo e cujo processamento é controlado pelo *Inputformat*.

- **RecordReader e *Input Split***: O ficheiro 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 ficheiros com formato não-divisíveis, como .gzip, o *input split* será igual ao tamanho do ficheiro.

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 registo pelo RecordReader. Algumas vezes o comportamento do RecordReader pode ser customizado, com a criação dum RecordReader proprio.

- **Mapper**: A classe *Mapper* é responsável pelo processamento do *input split*. A função *RecordReader* passa cada registo 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 registos na *input split* forem processados e, portanto, qualquer operação de limpeza deve ser executada dentro dele.

  O *Mapper* processa os registos 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 ficheiros de saída).
- **Partitioner:** O *Partitioner* atribui um número de partição ao registo emitido pelo *Mapper* para que os registos com a mesma chave obtenham sempre o mesmo número de partição, assegurando que os registos 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*.

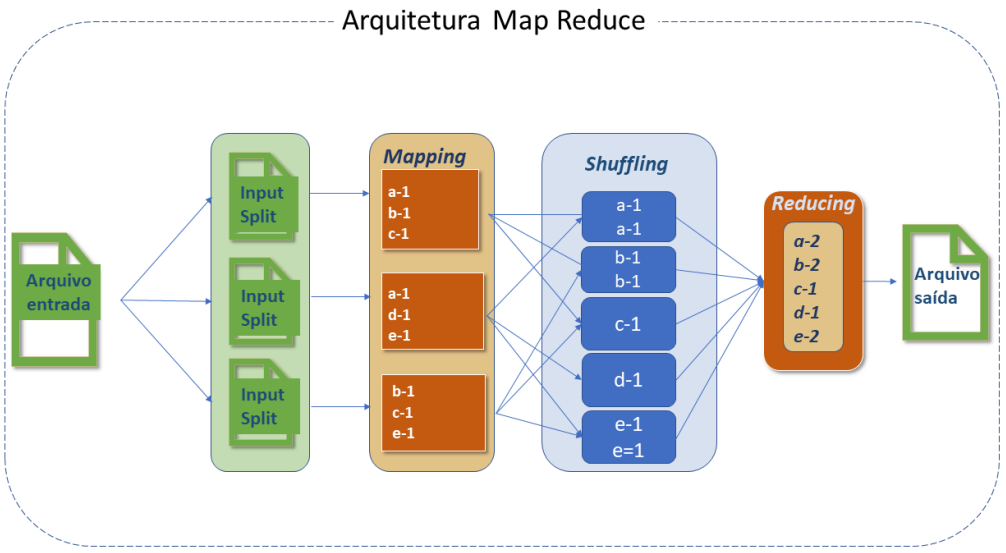
    - O número total de reduções geralmente é calculado como :

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

> **NOTA**
>
> 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 num ficheiro, no HDFS, por meio do *record writer* . Por default, os valores chave de saída são separados por *tab* e os registos são separados por um caracter *newline*. Isto pode ser modificado pelo usuário.



Arquitetura do Apache MapReduce

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 registos tok-k em seu nível e, em seguida, o redutor filtra os registos 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 numa 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 num *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 o desempenho de *jobs* e deve ser evitado a menos que seja

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

- *Configuração da CPU:* Na maioria dos Sistemas Operativos, 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 ficheiro `net.core.somaxconn=1024`.

- **Escolha do sistema de ficheiros:**

  - A distribuição Linux vem com um sistema de ficheiros 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 ficheiros padrão, que funciona melhor que o `EXT3`.
  - O Sistema de ficheiros regista o último horário de acesso para cada operação de leitura no ficheiro, e, portanto, causa uma escrita 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*.

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

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

- *Evitando formato de ficheiro não divisível:* O formato não divisível (p.ex: gzip) é processado de uma só vez.
  Se forem ficheiros muito pequenos, consumirá muito tempo, pois para cada ficheiro, 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 registos 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`.

- *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`.

> **AVISO**
>
> 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 ficheiro pelo *socket* TCP para o cliente.
    Na leitura *short circuit* o cliente lê diretamente o ficheiro, 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 escrita para os Datanodes inativos, a adição das propriedades abaixo no *hdfs-site.xml* é eficiente:

Terminal input

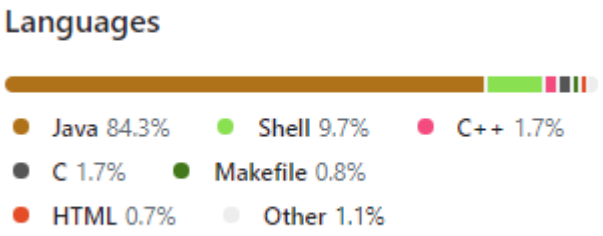
```
_dfs.namenode.avoid.read.stale.datanode=true
```

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

Apache YARN

Gerenciamento de Recursos

O Processamento de Big Data é uma tarefa difícil e não é viável num 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 partilhar 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.

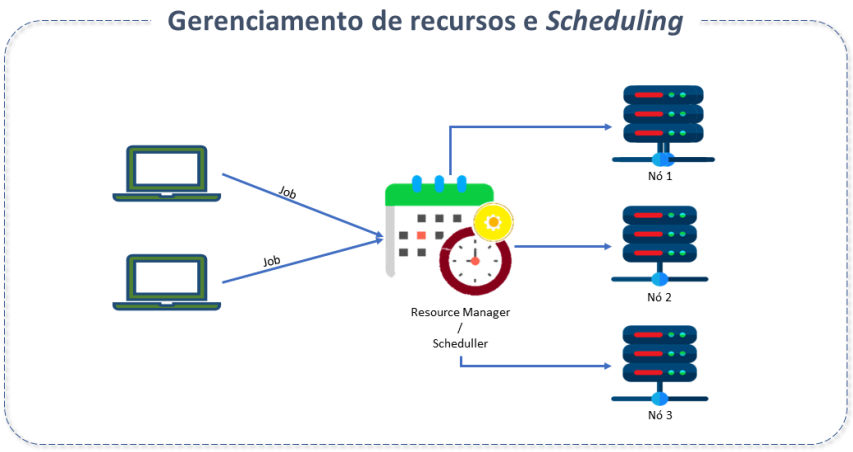
O Gerenciador de recursos age como um "scheduler", que "agenda" e prioriza solicitações, segundo os requisitos. Age, essencialmente, como um árbitro, que gerencia e aloca recursos disponíveis.

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.



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 disponibilizam um ambiente de execução personalizado no qual o código do aplicativo é executado, isolado do ambiente de execução do NodeManager e de outros aplicativos.

O Docker for YARN disponibiliza 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).

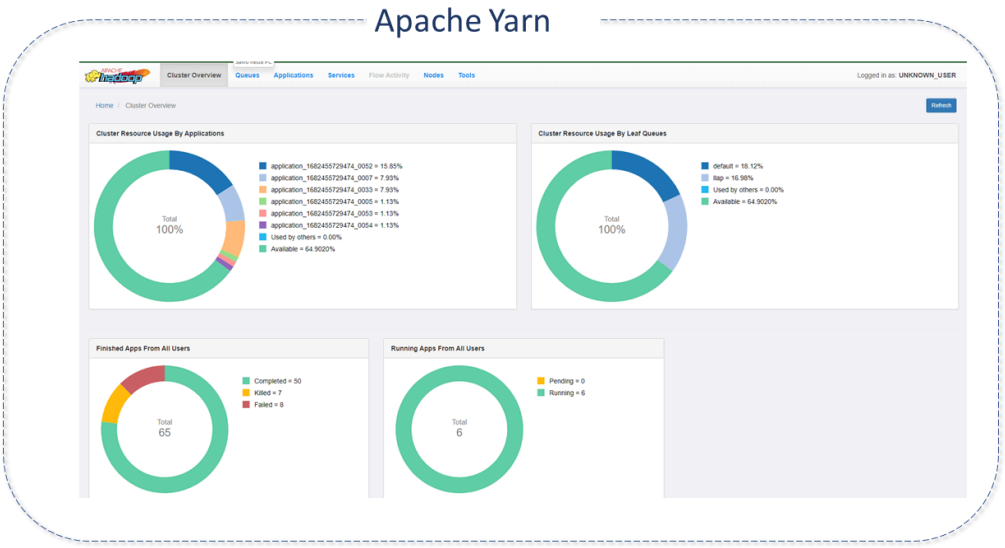
- **Alta disponibilidade para componentes:** A tolerância a falhas é um principio básico do Apache YARN. Essa responsabilidade é delegada ao Resource Manager(falhas de NodeManager e Application Master) e ApplicationMaster (falhas de *container*).

- **Modelo de Recursos flexíveis:** No Apache YARN, uma solicitação de recursos é definida em termos de memória, CPU, localidade, etc, resultando numa 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 num único ficheiro de log no HDFS.

- **Recursos eficientes e confiáveis:** o Apache YARN disponibiliza uma estrutura genérica de gerenciamento de recursos com suporte à análise de dados por meio de vários algoritmos de processamento de dados.



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 monitorização 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.

> **NOTA**
>
> *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 dum *Job* não ocorre no nó Master, mas num *container* instanciado numa 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 "conectável" 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 ordem 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 partilhar os recursos do *Cluster* de maneira econômica entre diferentes usuários. Em outras palavras, os recursos do *Cluster* são partilhados 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.

- 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 monitorizar *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 num 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 num algoritmo, a partir da "entrada" fornecida pelo cliente, a capacidade do Cluster e as filas e priorizações de recursos no *Cluster*.

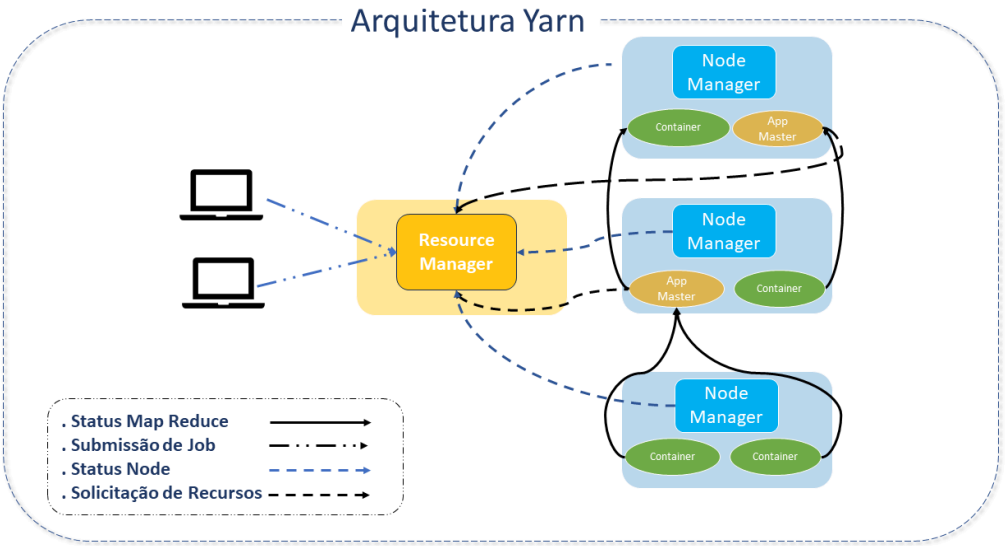
    Uma regra geral é iniciar o container no mesmo nó que os dados requeridos pelo job para facilitar a sua localização.

- **Application Master:** É uma biblioteca específica do framework, encarregada de negociar recursos do *ResourceManager* e trabalhar com o(s) *NodeManager(s)* para executar e monitorizar jobs. Ao receber um aplicativo, o ResourceManager iniciará o ApplicationMaster num container alocado, que se comunicará com o *Cluster* 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.

> **NOTA**
>
> 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*/monitorização 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

> **NOTA**
>
> Com a implementação do Apache YARN na versão 2.0 do Hadoop, o MapReduce mantém a compatibilidade com a versão estável anterior (Hadoop versão 1.x). Todas as tarefas do MapReduce continuam a ser 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 dum único perfil, semelhante aos tipos de instância Amazon Web Services Elastic Compute Cluster. É uma forma fácil de requisitar recursos a partir dum 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 controlo 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, ainda, informações e estatísticas sobre a instância NameNode e estatísticas sobre aplicativos e *containers*. Estes serviços podem ser usados por aplicativos de monitorização 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 controlo 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 disponibiliza o *ZKResourceManager StateStore* baseado no Zookeeper, que permite apenas um único *Resource Manager* gravando por vez.

    - **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 controlo. 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 dum *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 disponibiliza 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 partilhamento 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 partilhados 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 dum 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.

- 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:

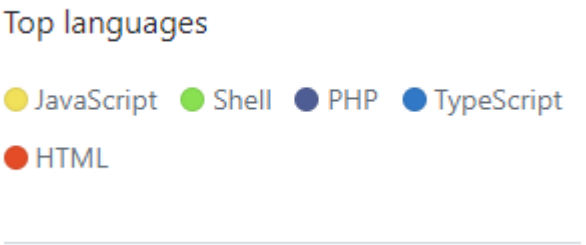
  - Persistência de Informações específicas do aplicativo:

- 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

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 desempenho 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 dum 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

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 fluxos de dados ilimitados e limitados. Foi criado pelo time 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 numa 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.

Arquitetura do Apache Flink

A arquitetura do Apache Flink é composta por vários componentes:

- ***JobManager***: Coordena a distribuição de tarefas e a gerenciamento 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 dum 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.

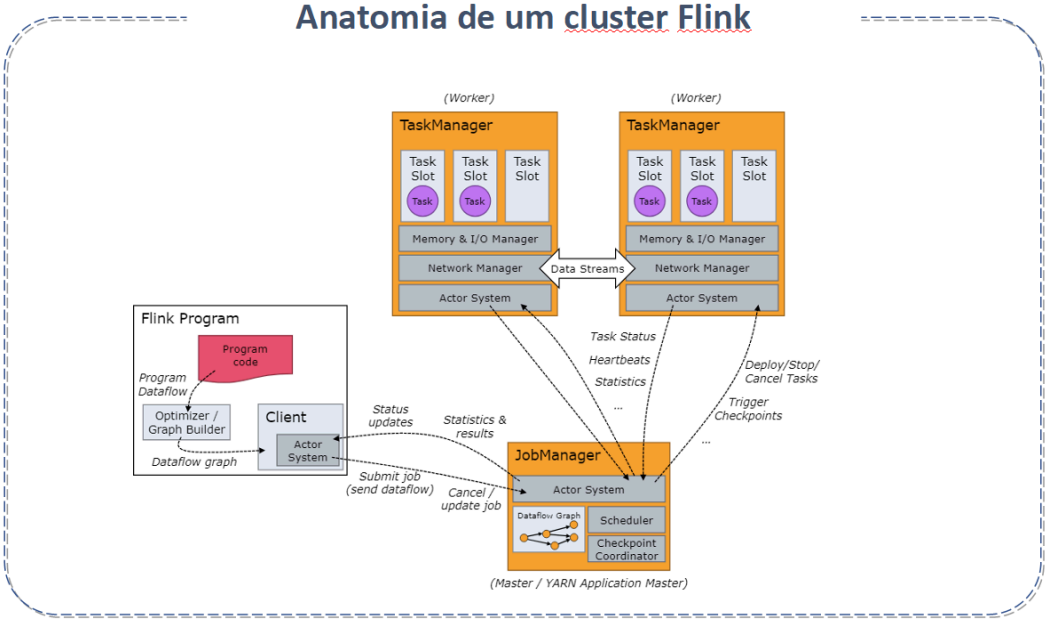


Figura 1 - Anatomia dum _Cluster_ Flink

Recursos do Apache Flink

- **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.
- **Alto Desempenho**: 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

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 monitorização 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**: *Apache Flink* oferece processamento de dados em lote e em fluxo; *Storm* é ótimo para processamento de fluxo.
- **Transformação de Dados**: *Apache Flink* tem opções ricas para transformação de dados; Storm é mais limitado.
- **Suporte a Aprendizado de Máquina**: *Apache Flink* suporta o aprendizado de Máquina, enquanto o *Apache Storm* não.
- **Linguagem de Consulta**: Apache 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*

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

- **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

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 dum processo cuidadosamente planeado para garantir que os dados recolhidos ou tratados cumprem normas, critérios e formatos previamente definidos.

Antes de passar para modelos de análise, relatórios ou aprendizado de máquina, 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 regulamentadas, até mesmo implicações legais.

A importância desse processo vai além da simples identificação de erros. É a base que garante a confiança organizacional nos 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, automatizar o processo de validação reduz a intervenção manual, tornando-a mais eficiente e evitando gargalos em pipelines complexos. Quando feito corretamente, também garante a conformidade com requisitos regulatórios e políticas internas, indispensáveis em setores como saúde, finanças e tecnologia.

Figura 1 - Garantia de qualidade

Características do Great Expectations

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

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

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

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

Arquitetura do Great Expectations

A arquitetura da Great Expectations foi projetada para ser modular e adaptável. É composto por três elementos principais:

- **Expectativas**: Representam as regras que os dados devem cumprir. Por exemplo, "os valores na coluna 'idade' devem estar entre 0 e 120."
- **Resultados da Validação**: Indique se os dados atendem ou não às expectativas, disponibilizando métricas detalhadas de conformidade.
- **Data Docs**: São relatórios interativos gerados automaticamente que disponibilizam 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 de dados e culmina na geração de relatórios que documentam e monitoram a qualidade dos dados ao longo do tempo.

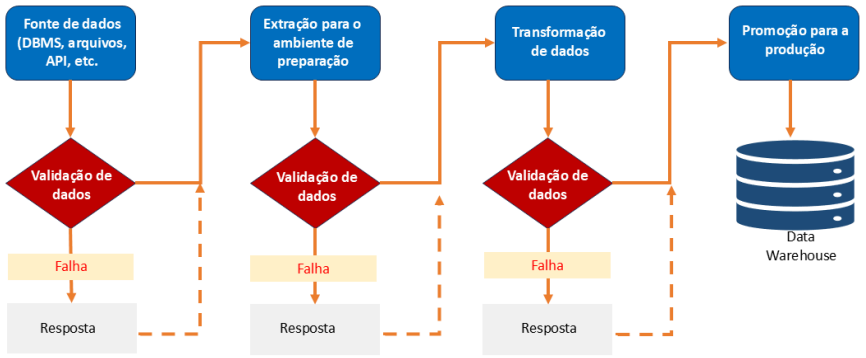


Figura 2 - Características do Great Expectations - Validação de Dados

Filosofia do Great Expectations

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

Formaliza estas expectativas num formato configurável, muitas vezes representado em JSON, permitindo uma fácil manutenção e integração com outras ferramentas.

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

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

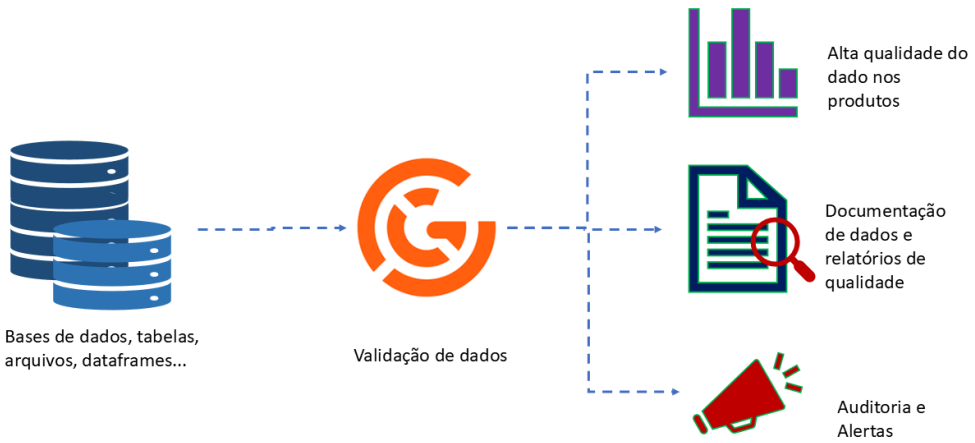


Figura 3 - Modelo de Great Expectations

Características 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**: Suporta ficheiros CSV, Parquet, bancos de dados relacionais e data lakes.
- **Local ou Cloud Execution**: Flexibilidade para executar em diferentes ambientes.
- **Configuração simples e intuitiva**: Interfaces amigáveis para definir e gerenciar expectativas.
- **Comunidade Ativa**: Atualizações constantes e suporte da comunidade.

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.

Detalhes do Projeto Great Expectations

O Great Expectations é construído em Python, aproveitando a popularidade e versatilidade da linguagem para facilitar a integração em pipelines de dados. Python é

amplamente utilizado por cientistas de dados e engenheiros, tornando o GX acessível e fácil de adotar.

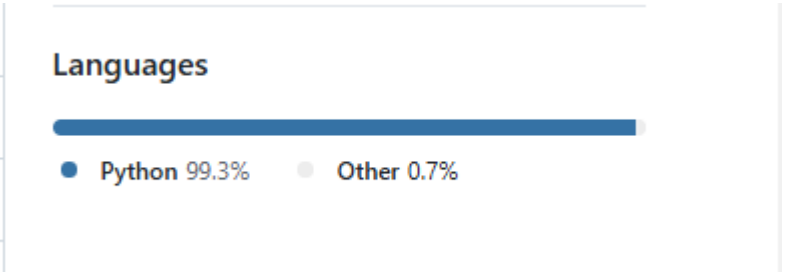


Figura 4 - Linguagem do Great Expectations

Fontes:

- Site Great Expectations

Apache HBase

Base de dados NoSQL

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

Estas bases 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.

As Bases de Dados NoSQL podem admitir linguagens de consulta similares à SQL.

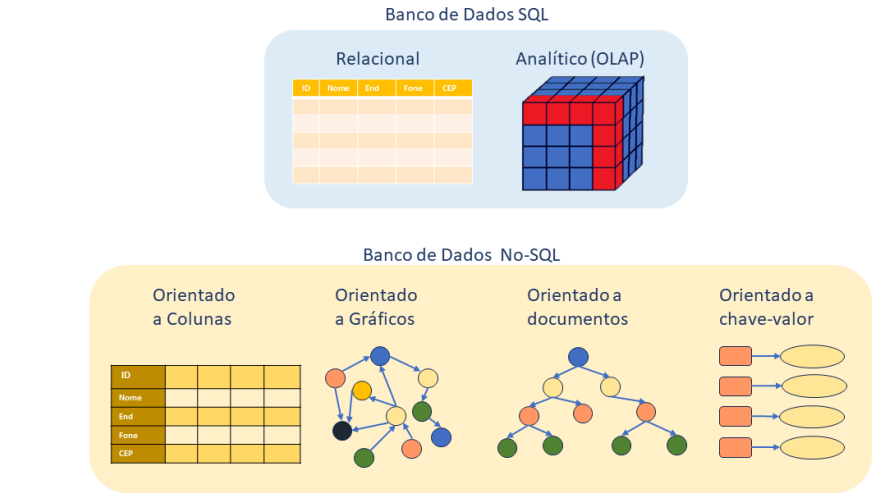


Figura 1 - Modelos de base de dados

Características do Apache HBase

HBase é uma Base de Dados distribuído, de código aberto, não-relacional ("NoSQL"), orientado a colunas, que executa sobre o HDFS, criado para disponibilizar acesso aleatório eficiente e leitura/escrita 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 Escrita 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 (pronta a usar) como seu sistema de ficheiros 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.
- **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 visibilidade operacional e métricas JMX.
- **Alta disponibilidade:** HBase provê alta disponibilidade com:
    - Informações de topologia de *Cluster* altamente disponíveis através de implementações de produção com múltiplas instâncias *HMaster* e *Zookeeper*.
    - Distribuição de dados em vários nós garantindo que a perda dum único nó afete somente os dados armazenados nesse nó.
    - HBase HA permite o armazenamento de dados, garantindo que a perda dum 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.

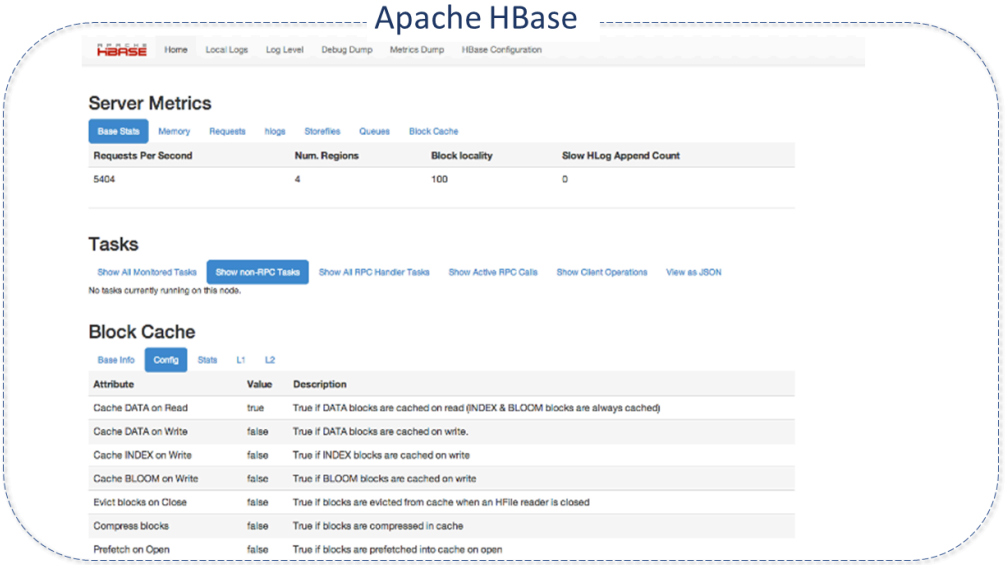


Figura 2 - Interface HBase

Arquitetura do Apache HBase

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

- **HMaster (Servidor mestre):**: é um processo responsável por designar regiões para Region Servers no *Cluster* Hadoop para balanceamento de carga. É responsável pela monitorização 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 num servidor dedicado, dentro do *Cluster* Hadoop.
  Existem outros *masters* em *standby* disponíveis para substituição em caso de indisponibilidade e o Zookeeper cuida disso.

  > **NOTA**
  >
  > 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 monitorização do *Cluster* Hadoop;
  > - Controlo 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 num único RegionServer. São responsáveis pela comunicação cliente e por executar todas as operações relacionadas a dados. Qualquer requisição de leitura ou escrita 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(registo de escrita antecipada) regista todas as alterações de dados do HBase. Numa 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 escrita no WAL falhar, toda a operação falha.
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 desempenho.
  O *Block Cache* segue o conceito *least recently used(LRU)*, em que registos 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 ficheiros 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/escrita* para esta região.
  Como regiões são blocos básicos de escalabilidade no HBase, é recomendado um baixo número de regiões por *Region Server* para garantir alto desempenho. 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 partilhar ou permutar com regiões, precisa abordar o Zookeeper antes.
  O serviço HMaster e todos os *Region Servers* são registados com o serviço ZooKeeper.
  O Cliente acessa o ZooKeeper para conectar-se com *Region Servers* e *HMaster*. Em caso de falha num 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 controlo de falhas de servidor e partições de rede, informações de configuração, etc.

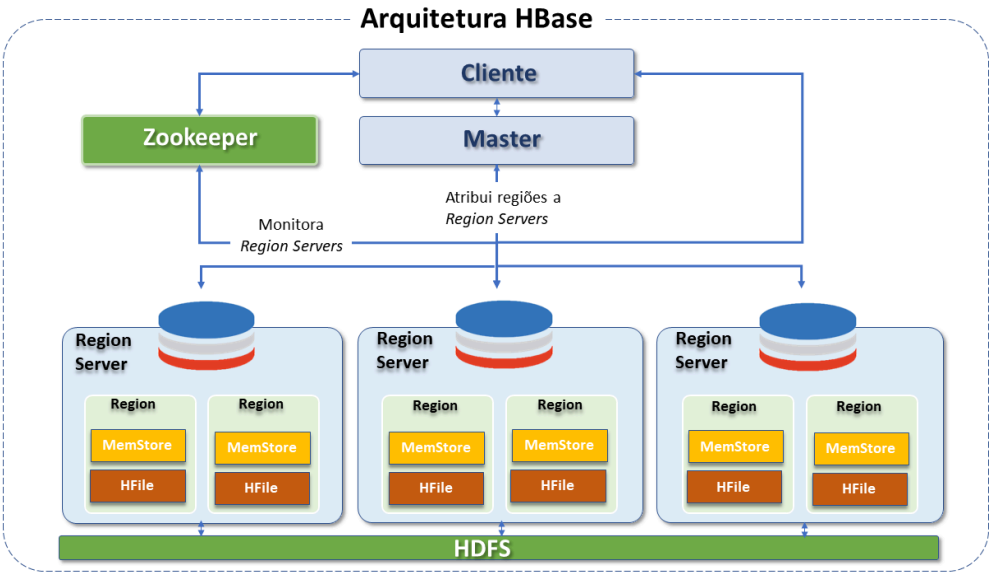


Figura 3 - Arquitetura HBase

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 contacto diretamente com o RegionServer sem necessidade dum

"coordenador central".

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

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 numa 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 na Base 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 ficheiro(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 numa tabela normal. Um qualificador de colunas é adicionado

a uma *column family* para disponibilizar 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.

- **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 dum agrupamento lógico de tabelas análogas a uma Base 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 gerenciamento de segurança para "locatários".

- **Grupos de Region Servers:** Uma tabela/namespace pode ser fixado(a) num subset de RegionServers, garantindo um certo nivel de isolamento.

- **Gestão da Segurança do Namespace:** Um Namespace/table pode ser fixado num 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*.

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 a Base 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 numa 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.

Recursos do Apache HBase

- **Segurança da Interface para o Usuário Web:** HBase provê mecanismos para assegurar vários componentes e aspectos e na sua relação 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 ficheiros 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 - controlo de Acesso baseado em papéis): Onde usuários e grupos podem ler e gravar num recurso específico ou executar um *endpoint coprocessor*, usando o paradigma de papéis.

    - Labels de visibilidade que viabilizam a rotulação de células e controlo 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 ficheiros subjacente, tanto HFiles quanto Wal, protegendo dados em repouso de uma invasão.

- **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 escrita "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 escrita 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 disponibilizam 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 num 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 ficheiros distribuídos adequado para armazenamento de grandes ficheiros. Entretanto, não é um sistema de ficheiros de uso geral e não disponibiliza pesquisas rápidas de registos individuais em ficheiros.

O HBase, por sua vez, é construído sobre o HDFS e disponibiliza pesquisas rápidas de registos (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 num algoritmo. É uma boa prática definir o número de regiões para a tabela e isto auxilia, inclusive, na melhoria de desempenho .

- **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.

- **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 num significativo decremento no desempenho.

- **Evitar compactação "maior", ou completa** HBase possui duas formas de compactação: a "menor", que combina um número configurável de *Hfiles* pequenos num maior e a compactação "maior", quando todos os ficheiros do *store* são lidos para uma região e gravados num único ficheiro. 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.

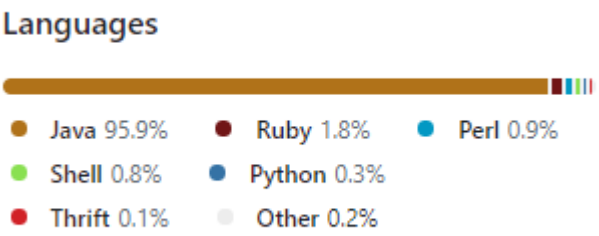


Figura 4 - Linguagens do HBase

Fontes

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

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

Apache Hive

Analytics

Uma Base 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.

Uma Base de Dados Analítica é um tipo de Base de Dados criada para armazenar, gerenciar e consumir dados, projetada para soluções específicas de Análise de Negócio, Big Data e BI.

Ao contrário de uma base de dados típica, que armazena dados por transações ou processo, uma Base de Dados analítica armazena dados históricos, em métricas de negócio. É baseada 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 bases de dados convencionais.

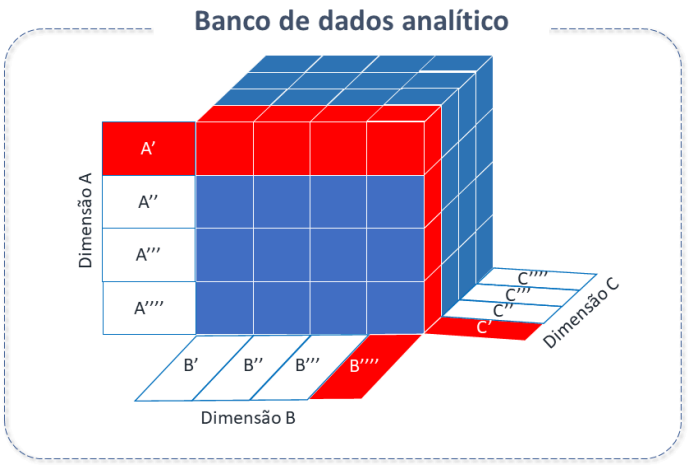


Figura 1 - Base de Dados Analítica

Características do Apache Hive

O *Apache Hive* é uma Base de Dados analítica (*Datawarehouse*) baseada no APACHE Hadoop, criada para facilitar o trabalho dos usuários data-warehouse com grande conhecimento em *queries* SQL, mas que encontram dificuldades para adotar Java ou outras linguagens.

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

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

Com o Apache Hive, 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:

    - Ficheiros 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 ficheiro 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 ficheiros 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, a ser usado em muitos projetos por todo o mundo. HQL provê comandos de *explain* e *analyze* que podem ser usados para checar e identificar o desempenho de queries. Adicionalmente, os logs do Hive contêm informação detalhada para investigação de desempenho e solução de problemas.

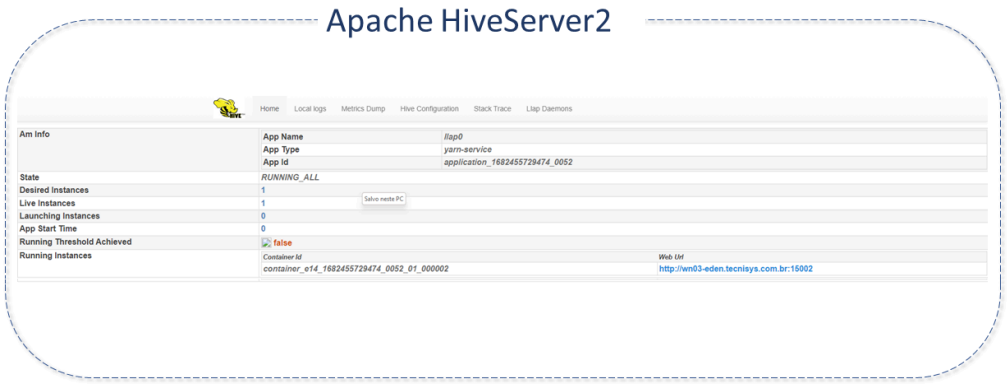


Figura 3 - Interface HiveServer2

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*.
- **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 monitoriza 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 dum novo plano DAG otimizado. A otimização pode ser feita em *joins*, *reducing suffling data*, etc., visando otimização do desempenho.

- ***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 ficheiro. O *executor* cuida do ciclo de vida das tarefas e monitoriza sua execução.

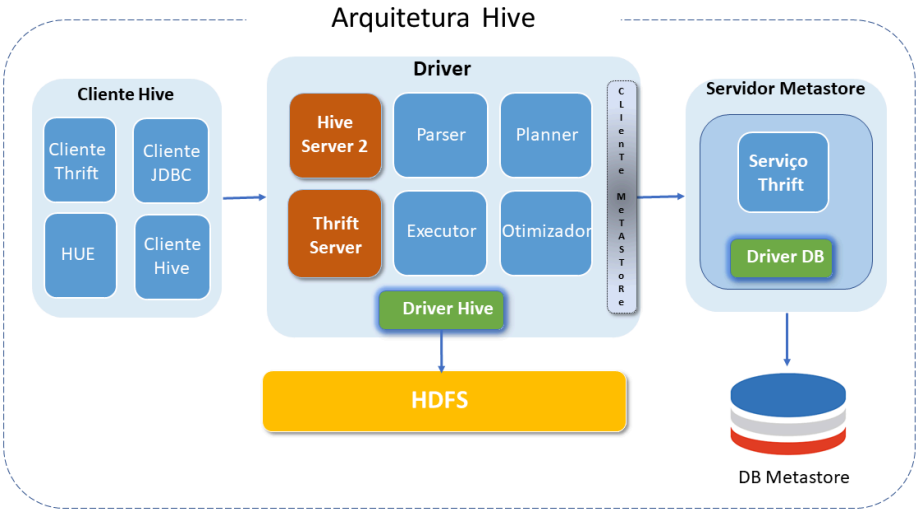


Figura 4 - Arquitetura Hive

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 ficheiros HDFS que são passadas pela árvore do operador associado).

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

Os ficheiros temporários são usados para disponibilizar dados para mapear/reduzir etapas subsequentes do plano.

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

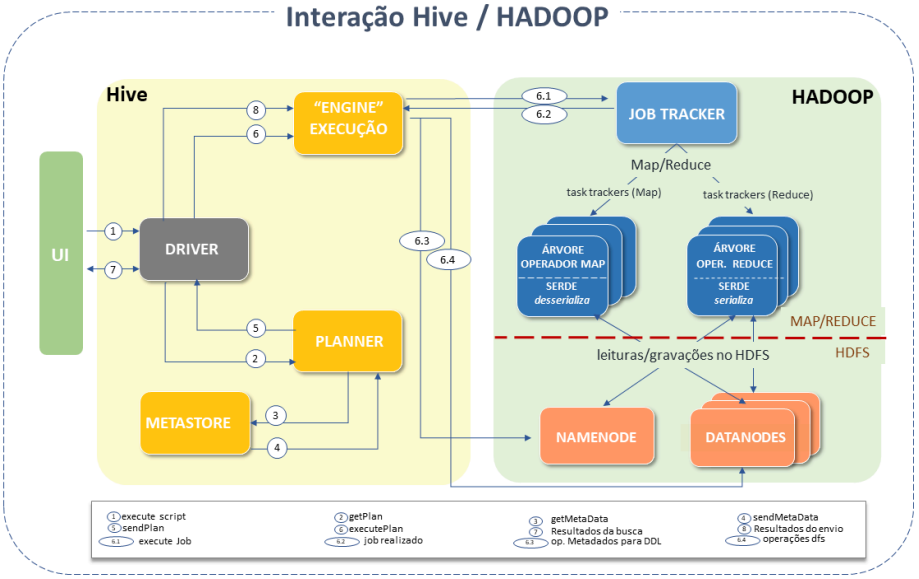


Figura 5 - Interação Hive-Hadoop

Como o Apache Hive Funciona

- **HCatalog**: 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 ficheiros 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 ficheiro).

  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.

  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** disponibiliza 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 numa Base de Dados Apache Derby embutido. Outras bases de dados do tipo cliente-servidor podem ser usados opcionalmente, como o Postgree.

Os primeiros quatro formatos de ficheiros suportados pelo Hive eram apenas texto simples, ficheiros 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:

- **Bases 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.

Recursos do Apache Hive

Construído sobre o Apache Hadoop, o Hive disponibiliza 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 disponibilizar 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** num SGBD relacional com acesso a estas informações por meio de API do serviço MetaStore.
- **Mecanismo para estabelecer estrutura numa variedade de formato de dados**.

- **Otimizador de consultas baseado em custos (CBO)**.
- **Acesso a ficheiros 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

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 escrita".

O Hive não verifica os dados em relação ao esquema da tabela na escrita. 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 disponibilizar 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 ficheiros recém-criados no Hive são ditadas pelo HDFS. O modelo de autorização do sistema de ficheiros distribuídos do Hadoop usa três entidades: usuário, grupo e outras com três permissões: ler, gravar, executar. As permissões padrão para ficheiros recém-criados podem ser configuradas alterando o valor unmask para a variável de configuração Hive hive.files.umask.value.

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 registo(s) da tabela HDFS (como sub-registo(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.

> **NOTA**
>
> 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 intermédios 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 ficheiros compactados com gzip não podem ser divididos. Significa que deve ser aplicado com cautela.

    - O tamanho do ficheiro 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.

- ***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 num 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, ficheiros 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 numa eficiente utilização da CPU.
    - Para ativar a vetorização, o parâmetro adequado é `set hive.vectorized.execution.enabled=true`.
- **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 registos de cada input split.

- Uma UDF pode ser implementada para filtrar os registos 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 escrita na saída padrão).

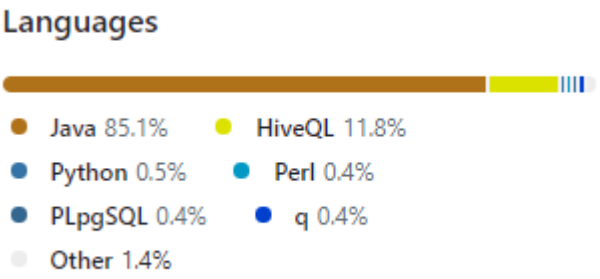


Figura 6 - Linguagens do Hive

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão actual 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

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 ficheiro nativos, servindo como uma camada de metadados e disponibilizando 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 registar dados transacionais com eficiência, a escalabilidade e a capacidade de atualizar ou excluir registos.

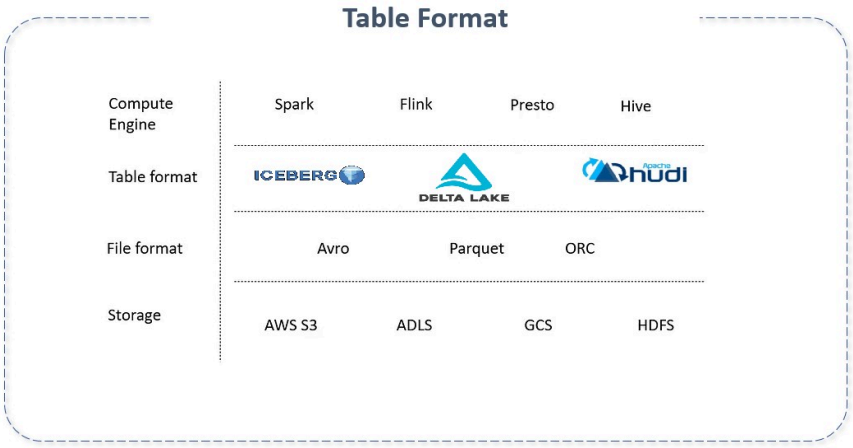


Table Formats adicionam uma camada de abstração aos ficheiros de dados

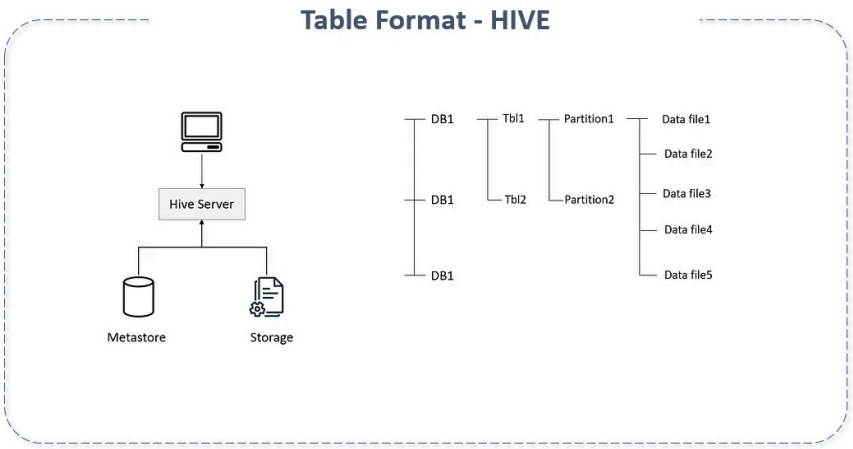
Os *table formats* evoluíram como resposta a uma necessidade crítica: combinar as vantagens de gerenciamento de dados de "armazéns" (Bases de Dados OLAP) com

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 ficheiros de dados. Tentam trazer características de bases 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 ficheiros é uma coleção, estamos falando em *table formats*. Quando um ficheiro é visto em formato de tabela, o diretório é visto como uma tabela, o que facilita seu rastreamento e o de seus ficheiros. Isto é feito com a ajuda de sistemas de catálogo. Sob ele, ficheiros são escritos em versões e aqueles mais recentes, correspondentes a uma tabela, são rastreados com a ajuda do metastore. O metastore é acedido para saber o estado atual da tabela e quais os ficheiros relacionados.

O Apache Hive é um dos mais antigos *table formats* usados. 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

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 ficheiros 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 coesa, 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 alto desempenho que funciona como uma tabela SQL.

Dentre suas principais características citamos:

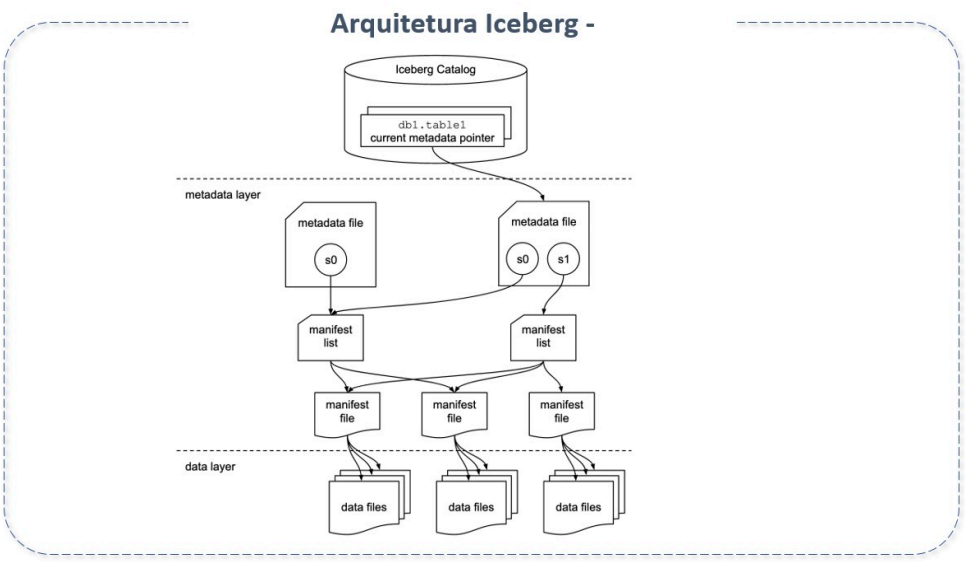
- **Evolução de esquema:** Suporta a adição, exclusão, atualização ou renomeação, sem efeitos colaterais;

- **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 ficheiro 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 ficheiro de metadados atual de cada tabela. O valor do ponteiro é o local do ficheiro de metadados).

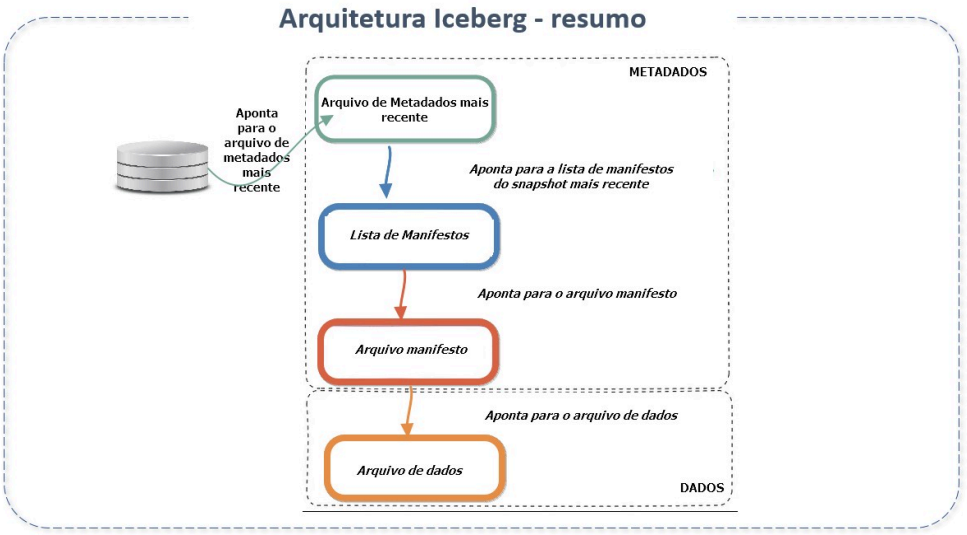
- **Camada de Metadados:** Os metadados são rastreados em três ficheiros, os ficheiros de metadados, a lista de manifestos e os ficheiros de manifesto.

- Ficheiros de metadados (*metadata file*): ficheiro .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 ficheiro de metadados e substitui o antigo com um *swap* atômico. Possui as seguintes subseções:

    - *Snapshot*: É uma lista completa de todos os ficheiros na tabela do *snapshot*. Representa o estado de uma tabela em determinado momento e é usado para aceder ao conjunto completo de ficheiros 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*): Ficheiro do tipo *avro* que contém todos os manifestos e suas métricas. Armazenam os metadados sobre os *manifestos* que compõnum *snapshot*, incluindo estatísticas de partição e contagens de ficheiros 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*.

  - Ficheiros de manifesto (*manifest file*):: Mantém o controlo sobre todos os ficheiros de dados, juntamente com detalhes e estatísticas como seus formatos, localização e métricas.
- **Camada de Dados**

  - Ficheiro de dados: O ficheiro físico real. Onde de fato estão os dados.



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 numa 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 ficheiros 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 controlo 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

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 registo é atualizado e um novo *snapshot* é criado, refletindo a atualização mais recente;
    - Com os *Commits* atômicos, garante que os dados permaneçam coesos 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 ficheiro, diferentemente dos catálogos tradicionais, que rastreiam registos por posição ou nome, o que exige leitura de diretórios e partições antes da atualização dum único registo. No Apache Iceberg é possível direcionar diretamente um único registo e qualquer atualização do registo sem nenhuma alteração na pasta. Isto porque os registos estão armazenados em seus metadados.
- **Recursos de desempenho**:
    - Cada ficheiro que pertence a uma tabela possui metadados armazenados para qualquer transação ocorrida, junto com estatísticas extras. Isto permite que usuários localizem apenas os ficheiros que correspondem à consulta que desejam no momento;
    - O Iceberg utiliza dois níveis de metadados para rastrear os ficheiros num *snapshot* - os ficheiros de manifesto e a lista de manifestos. O primeiro nível contém os ficheiros de dados, junto com os seus dados de partição e

estatísticas 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 ficheiro de dados. Assim, é possível planear sem ler todos os ficheiros de manifesto, usando a lista de manifestos para restringir o número de ficheiros de manifestos necessários à leitura.

Com isto, é possível alcançar consultas eficientes e econômicas em ficheiros.

Boas Práticas no Uso do Apache Iceberg

O Apache Iceberg é uma ferramenta poderosa para gerenciamento de grandes conjuntos de dados analíticos. Para maximizar a 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;

- **Gestão de Metadados**: Mantenha os metadados limpos e atualizados. Isto inclui remover *snapshots* antigos e ficheiros 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;

- **Monitorização e Optimização**: Monitore regularmente o desempenho das consultas e otimize as tabelas conforme necessário. Isto 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;

- **Segurança e Controlo de Acesso**: Implemente controlos 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 o gerenciamento 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 robusta 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;

- **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 no desempenho de Leitura**: Para melhorar o desempenho da leitura através de indexação e otimizações de ficheiro;

Estes itens destacam cenários ideais para a implementação do Apache Iceberg, maximizando os 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. Isto 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

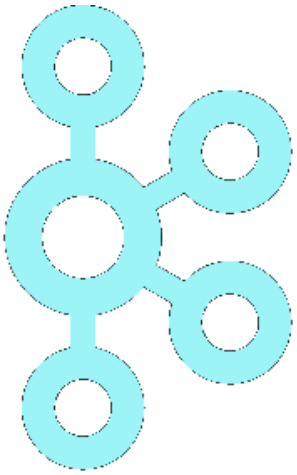
> **DISPONÍVEL NO TDP KUBERNETES**

> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão actual é **1.4.2**, distribuída pelo Helm Chart `tdp-iceberg` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fonte(s):

- Iceberg.apache.org

Apache Kafka



Streaming de Dados

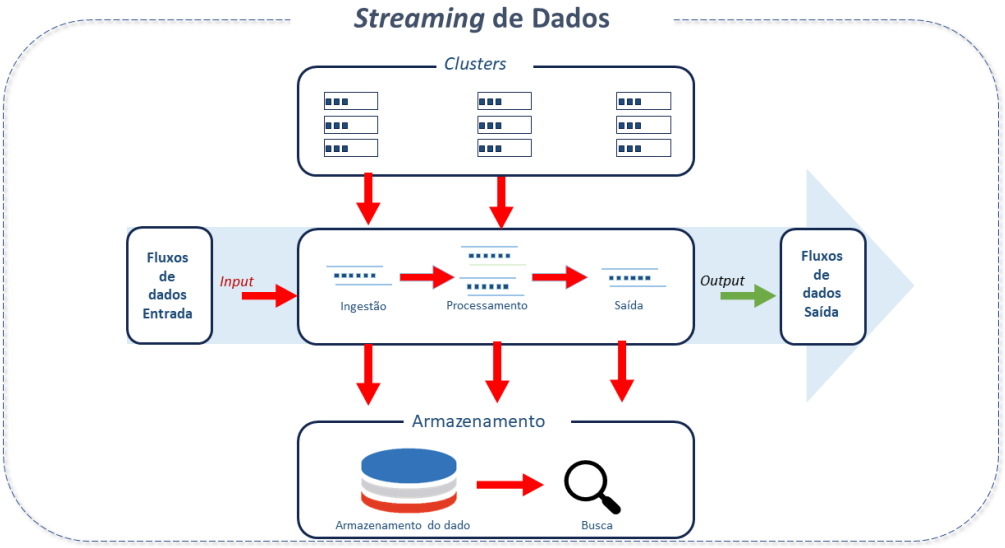
Métodos legados de processamento em lote exigem que os dados sejam coletados em forma de lote antes de serem processados, armazenados ou analisados.

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.



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 registo 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.

- Gerenciamento de diversos formatos e tipos de dados usando formatos de dados binários.

> **NOTA**
>
> 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 alto desempenho.
  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 ficheiros de *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 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* dum ou mais servidores que pode estender-se a múltiplos Datacenters ou *Cloud-Regions*.
  Alguns destes servidores, denominados *brokers*, formam a camada de armazenamento.

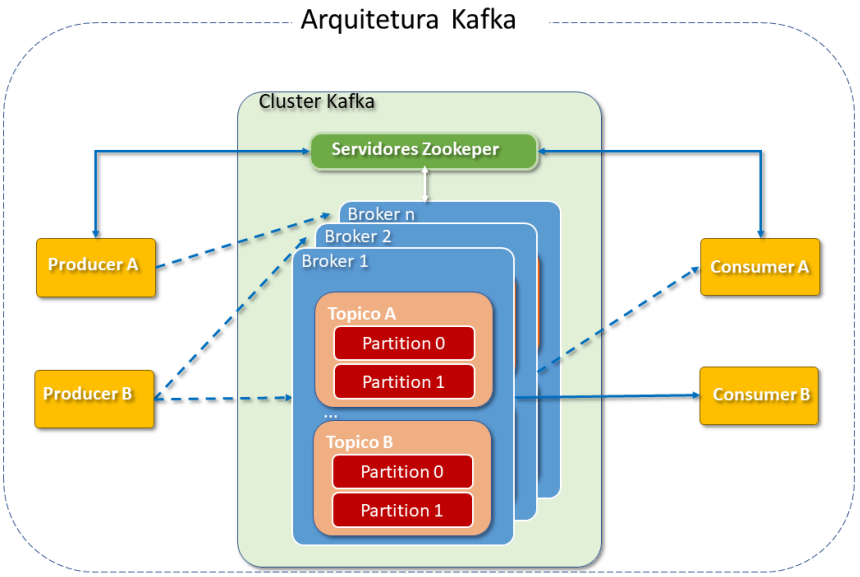
  - **Brokers:** Um *Cluster* Kafka tipico consiste de multiplos *brokers*. Isto ajuda no balanceamento de carga nas leituras e gravações no cluster.
    Cada broker é *steteless*(não permanece dedicado à conexão) e conta com o Zookeeper para manter seu estado.
- **Topics: (tópicos)** São a forma de organização dos eventos. Sua função é similar às tabelas dum 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 ordem em que chegam. Eventos com a mesma chave(id) são gravados na mesma partição.
  O número de partições dum tópico é configurável, assim como o seu tamanho.
  Contar com mais partições num 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 escrita para suas respectivas partições.
  Os seguidores replicam o lider mas não interferem no seu trabalho, atuando, como uma cópia de segurança.
  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.
  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, registando 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 escrita 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 ficheiros 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 monitorização de *clusters* Kafka.

Trata-se de uma ferramenta open source que auxilia na observação dos fluxos de dados, detecção e solução de problemas, gerenciamento 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** — monitorização e gerenciamento de todos os clusters num só lugar
- **Monitorização 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
- **Controlo de acesso baseado em função** - gerenciamento de permissões para aceder 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**:

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(escrita) de fluxos de eventos num ou mais topicos.
  - **Consumer-API**: Para subscrição(leitura) dum 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.
- **Monitorização Operacional**: Kafka é frequentemente usado para monitorização Operacional, que envolve estatísticas de aplicativos distribuídos para produção de fontes centralizadas de dados operativos.
- **Solução de Agregação de *Logs*** Kafka pode ser usado como Solução de Agregação de *Logs*, coletando ficheiros de *log* físicos de servidores e colocando-os num 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

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 partilha 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 uma percentagem da rede e dos *threads* de E/S (a partir de 0,11).

Boas Práticas para Apache Kafka

- **Validação de dados:** Durante a escrita dum 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 escrita dum *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

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 dum 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.

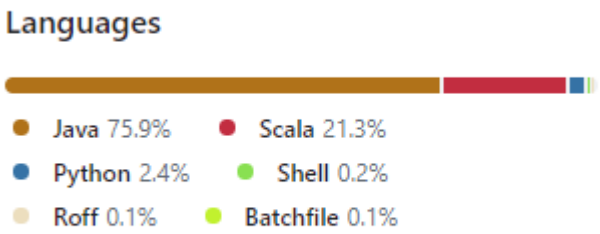
- **Commit offsets na hora certa:** No caso de *commit offset for messages* é necessário fazê-lo no momento correto. Um aplicativo com processamento `em lote` toma mais tempo para completar o processamento.
  Não é uma regra, mas se o processamento durar mais que um minuto, é razoável realizar o *commit 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

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

Fonte(s):

TDP Kubernetes

> **DISPONÍVEL NO TDP KUBERNETES**
>
> Este componente também está disponível na edição **TDP Kubernetes** desde a versão 3.0.
>
> A versão actual 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

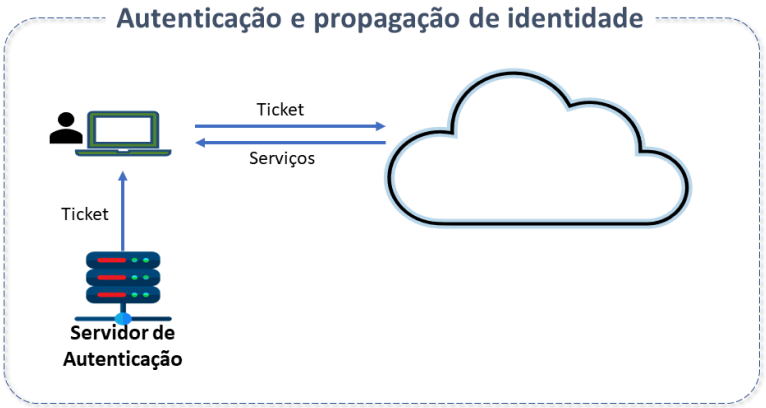
Kerberos

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 dum 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 disponibilizar 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 dum 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 disponibiliza Login Único (SSO) baseado

num serviço de autenticação mútua confiável (usuários e serviços dependem de uma terceira parte: o servidor Kerberos). Foi adotado pelo time 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 cópia de segurança 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**: As senhas ou chaves secretas são conhecidas apenas pelo Centro de Distribuição de Chaves (KDC) e pelos principais do Kerberos (identidades únicas, como utilizadores ou serviços), tornando o sistema escalável para autenticar um grande número de entidades. Cada entidade precisa apenas de conhecer a sua própria chave secreta e registá-la previamente no KDC.

- **Segurança**: A senha do usuário nunca é transmitida pela rede. O Kerberos utiliza bilhetes temporários (como o TGT e o Bilhete de Serviço), que são emitidos pelo servidor de autenticação (KDC) e têm um 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 o gerenciamento 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 fornecedores de identidade empresariais, como o Active Directory, o FreeIPA ou sistemas baseados em LDAP.

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 disponibiliza tickets válidos temporariamente ao usuário para aceder 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 dos consumidores, o Kerberos utiliza criptografia de chave simétrica (a mesma chave é usada para cifrar e decifrar os dados) e um componente central denominado KDC (Centro de Distribuição de Chaves), que mantém uma base de dados com todas as chaves secretas. Este processo envolve três componentes principais:

- **Servidor de Autenticação (AS)** – realiza a autenticação inicial do utilizador e emite o Ticket Granting Ticket (TGT).
- **Servidor de Concessão de Bilhetes (TGS)** – emite bilhetes específicos para serviços com base no TGT previamente obtido, evitando que a palavra-passe do utilizador tenha de ser reutilizada.
- **Base de Dados Kerberos** – armazena as chaves secretas e as identidades de todos os utilizadores e serviços autorizados no sistema.

A ideia que norteia a solução é a existência dum servidor capaz de entregar *tickets* ao usuário para aceder serviços. Esses *tickets* permanecem válidos por um determinado período. O serviço não necessita de contactar o KDC para validar o bilhete, pois consegue validá-lo localmente utilizando a chave secreta que partilha com o KDC.:

- O cliente solicita um *ticket* ao servidor Kerberos.
- O cliente submete o *ticket* ao serviço pretendido e é autenticado.

> **NOTA**

> 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*.

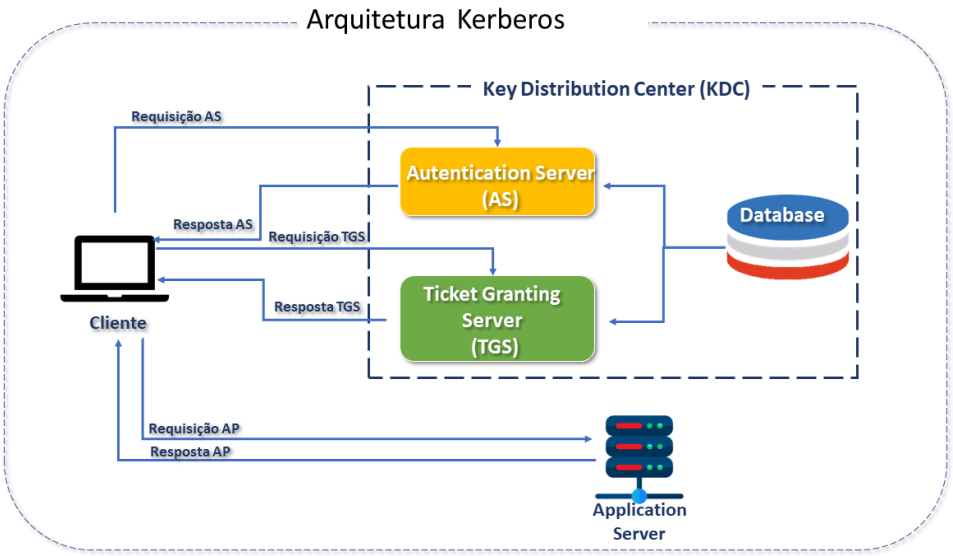
Após a autenticação, o KDC emite o *ticket*, que será inserido no conjunto de credenciais privadas. O serviço pode então atender ao cliente.

O processo de autenticação do usuário ocorre, resumidamente, da seguinte forma:

- O usuário se autentica usando seu Kerberos Principal.

> **NOTA**
>
> 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 ficheiros.



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 aceder 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 dum 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:

- Portal do MIT
- Wikipedia

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 disponibiliza 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 disponibiliza uma segurança reforçada.

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 dum 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, disponibilizando SSL/TLS *out-of-the-box* (pronto para uso).

O Knox enviará solicitações externas dos clientes Hadoop aos serviços correspondentes e, antes de enviar, disponibiliza todos os serviços de segurança configurados no *Cluster*.

- **Controlo 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.
- **Registo de auditoria.**

Arquitetura do Apache Knox

O Apache Knox age como um ponto único de contacto 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.

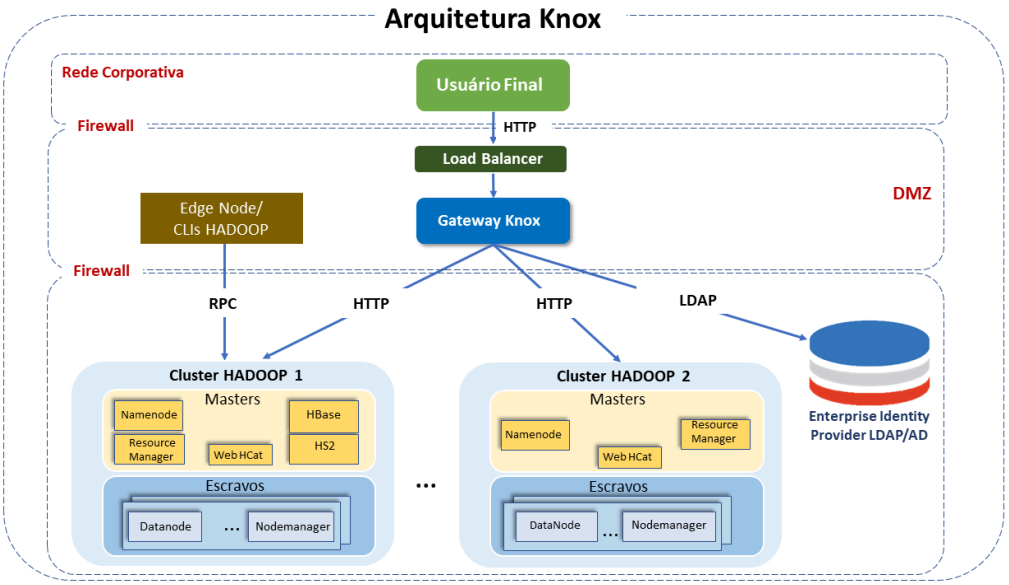
Sua principal característica é o provimento dum perímetro de segurança para APIs REST Hadoop, restringindo o número de *endpoints* de rede requeridos para aceder o *Cluster* Hadoop.

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*.

Numa solução empresarial que emprega *Clusters* protegidos por Kerberos, provê segurança que integra-se bem com soluções de gerenciamento 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:



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

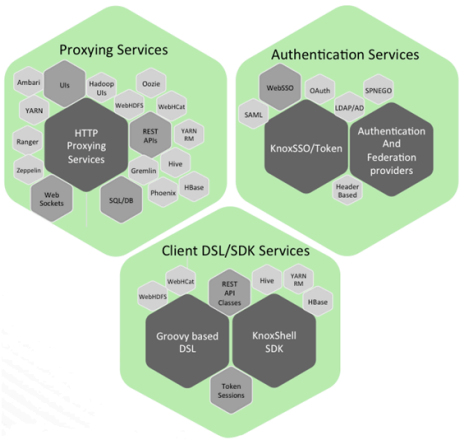
por meio de uma cadeia de provedores que são definidos no *topology deployment descriptor* (descritor de implementação de topologias) para cada *Cluster* Hadoop controlado pelo Knox. A definição de *Cluster* também é feita no descritor de implantação de topologia e disponibiliza ao Gateway Knox o layout 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 escrita 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*, disponibilizando 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

- **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 num 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 dum 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 disponibiliza um mecanismo de autenticação baseado em formulário que aproveita a autenticação Shiro para autenticar no LDAP/AD com credenciais coletadas dum 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

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 numa extensibilidade 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 aceder 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 ficheiro 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")

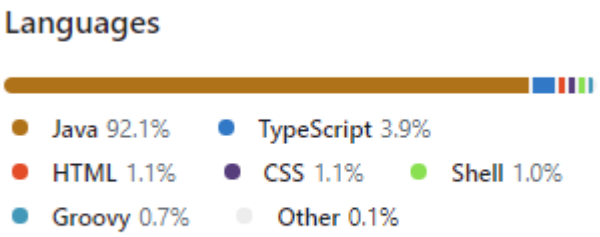
Componentes suportados por Knox

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

O Apache Knox foi desenvolvido em Java.



Linguagens do Knox

Fontes:

- Comunidade Apache Knox
- GitHub

Apache Livy

Gestão de Sessões Spark

Apache Livy é uma plataforma de serviço de código aberto que disponibiliza 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 envio programado 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.

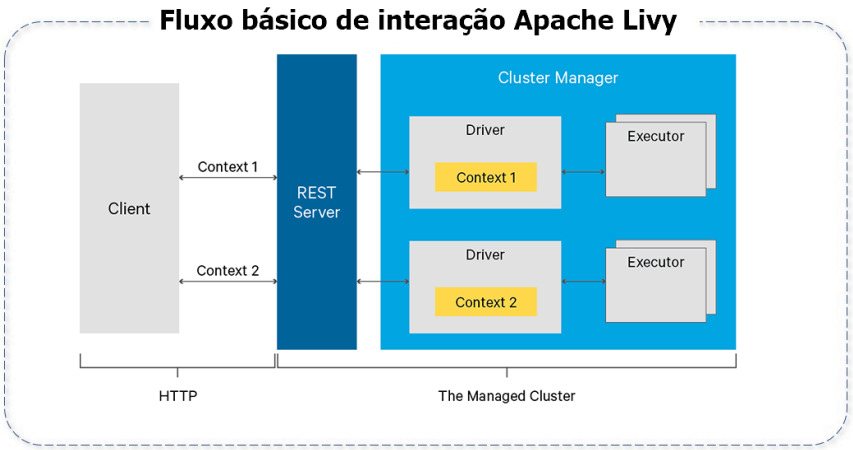
O Apache Livy foi inicialmente desenvolvido pela Cloudera como uma solução para simplificar a interação com *clusters* Spark. Desde o início, o foco do Livy tem sido oferecer uma interface RESTful para facilitar a submissão e o gerenciamento de trabalhos Spark em *clusters* de dados, especialmente em 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 Controlo de Acesso**: Inclui recursos para garantir a segurança dos dados e do acesso ao *cluster*;

- **Integração com Hadoop YARN**: Facilita o gerenciamento de recursos em *clusters* por meio da integração com YARN.

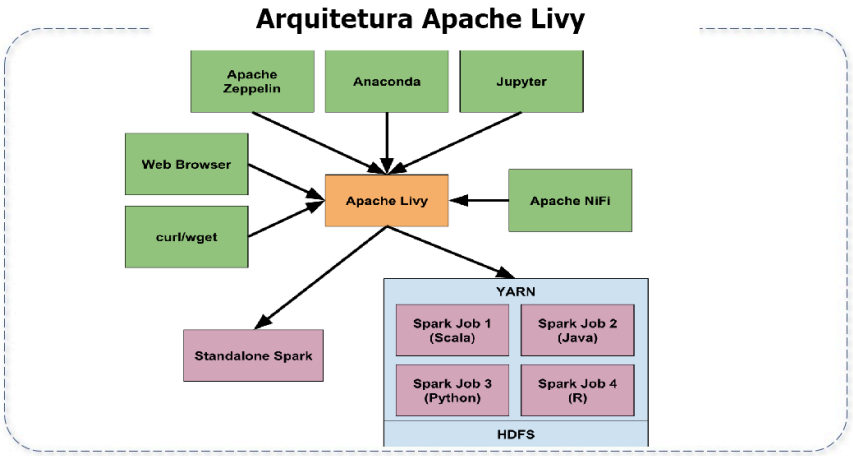


Fluxo Básico Apache Livy - submissão de _jobs_ por meio do Livy

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 controlo de trabalhos Spark.



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 controlos de segurança robustos;
- **Monitorização**: 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.

Fonte(s):

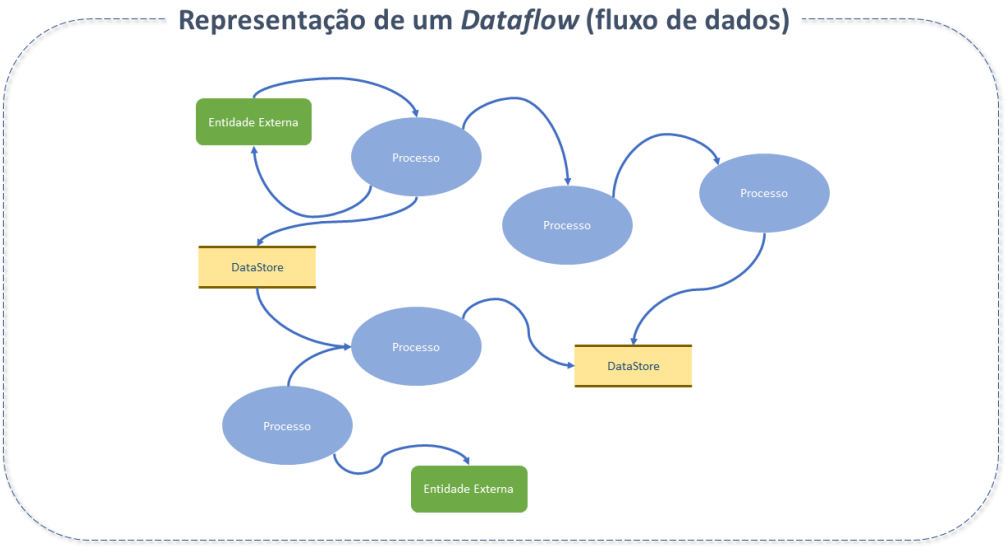
- Livy.apache.org

Apache NiFi

Automatização de Fluxo de Dados

Automatização de fluxos de dados é um paradigma de software baseado na ideia de computação como um grafo direcionado, representando o fluxo automatizado e gerenciado da informação entre sistemas.

A automatização de fluxos de dados auxilia na captura, construção e colaboração em escala através de ferramentas automatizadas, garantindo maior eficiência dos sistemas através de relatórios otimizados e controláveis.

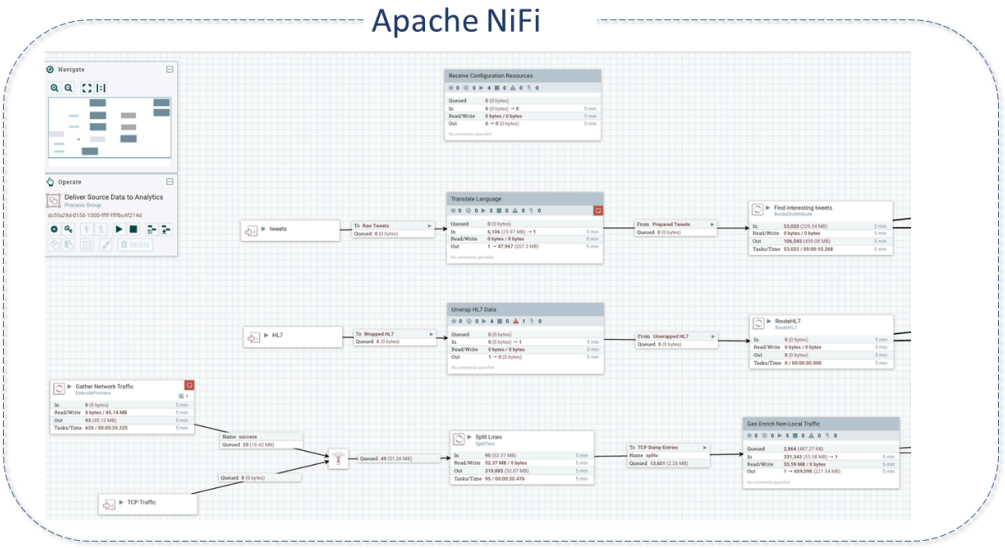


Representação dum Fluxo de dados

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.



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.

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:

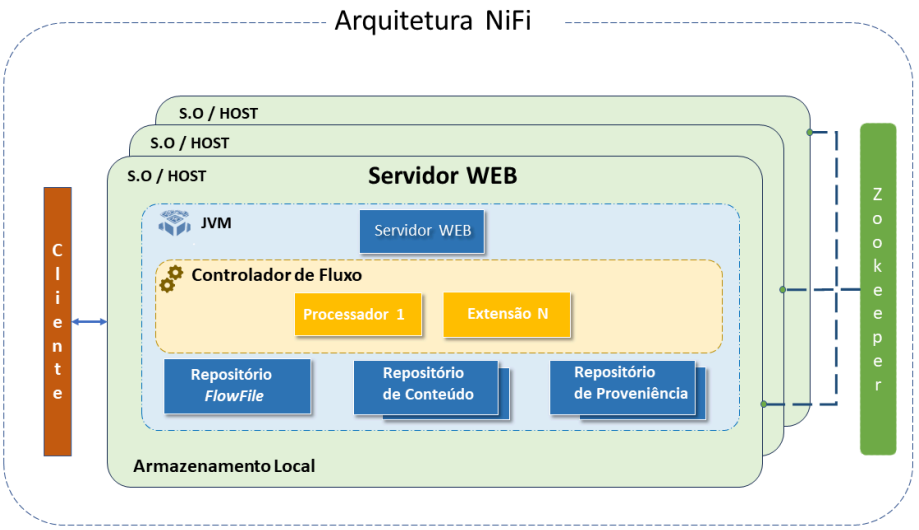
- A criação visual e gerenciamento de grafos direcionados (com sentidos associados à cada aresta) de processadores.
- Alta taxa de transferência e um *buffer* de dados natural, apesar das flutuações no processamento e nas taxas de fluxo.
- Modelos altamente concorrentes, com a capacidade de executar várias tarefas distintas simultaneamente (ou aparentemente simultâneas), permitindo que o desenvolvedor não se preocupe com as 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, num sistema operativo *host*. Seus principais componentes na JVM são:

- **Web Server (Servidor Web)**: executa o controlo visual e monitoriza os eventos. Hospeda o comando *HTTP-based* e a API de Controlo.

- **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 "extensibilidade" 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 monitorização e do estado interno do NiFi para *endpoints* como ficheiros de *log*, email e *web services* remotos.
  - **ParameterProvider**: Provê parâmetros a usar por fontes externas.
  - **ControllerService**: Fornece mecanismo para criar serviços partilhados entre todos os Processadores, *ReportingTasks* e outros *ControllerServices* numa única JVM.
  - **FlowFilePriorizer**: Fornece um mecanismo pelo qual os FlowFiles numa fila podem ser priorizados ou classificados, para posterior processamento numa 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 "conectável". O "default" é um *log write-ahead* persistente localizado numa partição do disco específica.
- **Repositório de Conteúdo**: É onde estão são mantidos os dados "em trânsito". É "conectável". 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 informações sobre a proveniência dos dados que circulam pelo sistema. O repositório é "conectável" com a implementação *default* a usar um ou mais volumes de discos físicos. Dentro de cada local o dado de evento é indexado e habilitado para busca.



Arquitetura do NiFi

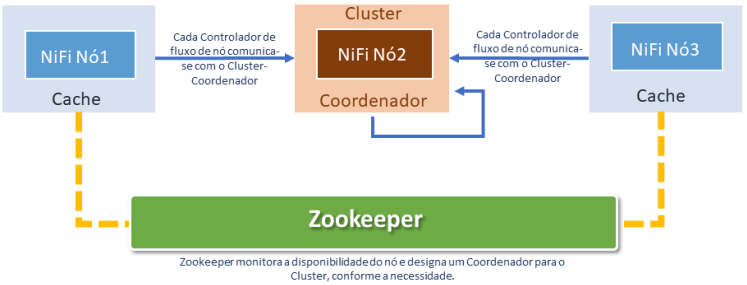
O NiFi também está habilitado a operar dentro dum *Cluster*.

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 num 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

Assim como o gerenciador de *Dataflow*, a interação com o *Cluster* NiFi pode se dar através 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 "conectáveis" para a maior parte dos subsistemas NiFi, o desempenho dependerá da implementação. Considere usar as implementações padrão prontas a usar.

> **DICA**
>
> Podemos assumir uma taxa de "*leitura/escrita*" de 50MB por segundo em discos RAID modestos dentro dum 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 Base de Dados ou credenciais de provedor de serviços em nuvem.

Os serviços do controlador são *daemons*(executados em segundo plano) e disponibilizam configuração, recursos e parâmetros para os processadores executarem.

> **DICA**

> 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 armazenamento 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.

> **DICA**
>
> *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 gerenciamento de recursos partilhados 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 controlo de versão e armazenados num registo. O registo organiza onde os fluxos são armazenados e gerencia as permissões para acessá-los, criá-los, modificá-los ou excluí-los.

A interface de Usuário(UI) do NiFi Registry exibe os recursos partilhados disponíveis e disponibiliza 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.

> **NOTA**
>
> *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 na gerenciamento de fluxo de dados, com foco na coleta de dados na origem da sua criação.

O Apache NiFi MiNiFi disponibiliza 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.

Gestão 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 escrita 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 disponibilizar *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

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 detalhamento da configuração de fluxo.

Facilidade de Uso

- **Comando e Controlo 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 detalhadas 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 partilhamento 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 regista, indexa e disponibiliza automaticamente dados de proveniência à medida que os objetos circulam 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 detalhado**: 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 dum 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 SSL bidirecional.

Adicionalmente, permite que o fluxo criptografe e descriptografe o conteúdo e use chaves partilhadas ou outros mecanismos em ambos os lados da equação remetente/destinatário.

- **User to System**: NiFi permite a autenticação SSL bidirecional e provê autorização "conectável" 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 dum *Dataflow* é aplicado a cada componente, viabilizando que usuário admin tenha um nível detalhado de controlo 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 gerenciamento 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

- **Extensibilidade**: NiFi é construído para extensibilidade, 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 extensibilidade 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 extensibilidade.
  O conceito destes pacotes de extensibilidade é chamado "ficheiros 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.

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*" (Gestão de Workflows).

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 controlo dum 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 dum "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 num 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:

- Pode tornar o processador muito complexo.
- Se o processador não conseguir se comunicar com um serviço remoto, encaminhará os dados para uma relação *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 intermédios são descartados quando o processador termina de enviar para o serviço remoto. O formato intermédio pode ser útil para outros processadores.

> **NOTA**
>
> 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 disponibilizar uma aparência consistente aos usuários, é aconselhável que os processadores mantenham as convenções de nomenclatura padrão:

- Processadores que extraem dados dum sistema remoto são denominados:
  - `Get<Service>` (pesquisam dados de fontes arbitrárias através de protocolo conhecido, como GetHTTP, GetFTP) ou
  - `Get<Protocol>` (pesquisam dados dum serviço conhecido como GetKafka).
- Processadores que enviam dados para um sistema remoto são denominados:
  - `Put<Service>` ou
  - `Put<Protocol>`.
- Nomes de relações são caixa baixa e usam espaço para delinear palavras.
- Nomes de propriedade devem usar palavras significativas, como o título dum livro.

Anotações de Comportamento do Processador

Ao criar um processador, o desenvolvedor deve disponibilizar 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 ficheiros *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 numa 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 alguma relação 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 numa fila de entrada.
- **InputRequirement**: ao disponibilizar 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

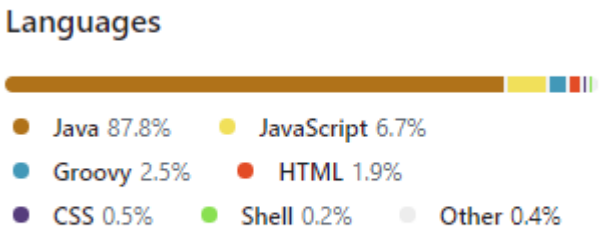
O NiFi disponibiliza uma capacidade genérica de processamento de dados. Os dados podem estar em qualquer formato.
Os processadores geralmente são escalonados com vários *threads*.
Um erro comum dos desenvolvedores é armazenar em *buffer* todo o conteúdo dum

FlowFile na memória.
Salvo em casos em que é absolutamente necessário, isso deve ser evitado, a menos que se conheça o formato dos dados.

> **DICA**
>
> 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 actual é **1.28.0**, distribuída pelo Helm Chart `tdp-nifi` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- NiFi.apache.org
- NiFi.apache.org/registry
- cwiki.apache.org

Apache Ozone

Object Storage

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

Ozone é um sistema de armazenamento de objetos redundante e distribuído, otimizado para cargas de trabalho de Big Data. O principal foco do design do Ozone é a escalabilidade, visando crescer até milhares de milhões de objetos.

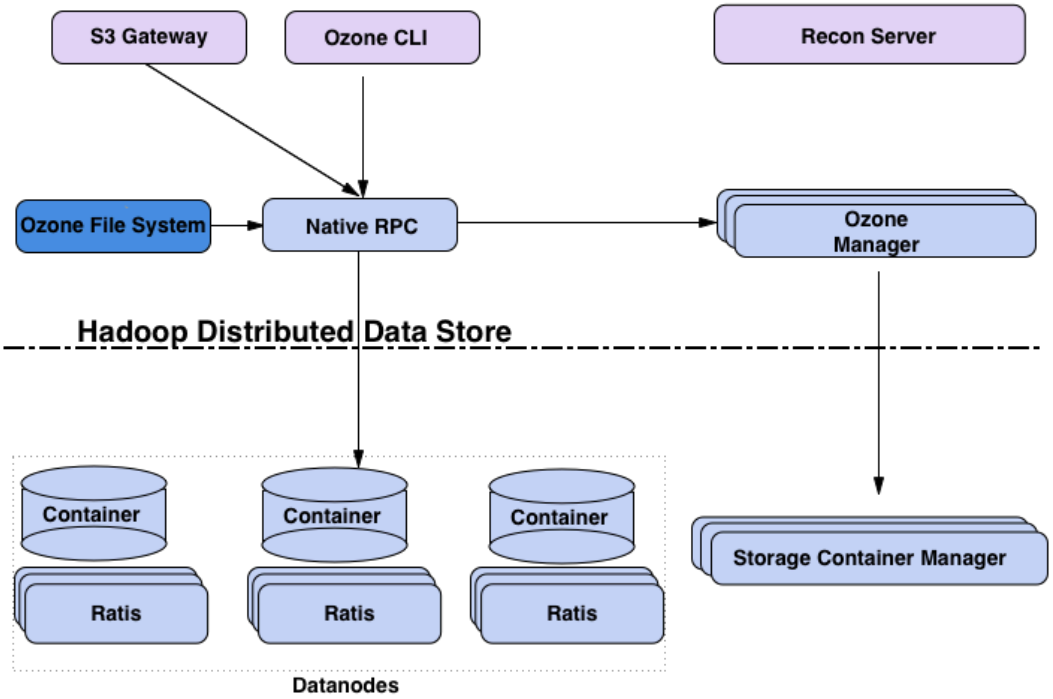
Ozone separa a gestão do namespace da gestão do espaço de blocos, o que lhe permite escalar de forma muito mais eficiente. O namespace é gerido por um daemon chamado Ozone Manager (OM), enquanto o espaço de blocos é gerido pelo Storage Container Manager (SCM).

Ozone é composto por volumes, buckets e chaves. Um volume é semelhante a um diretório pessoal no ecossistema Ozone. Apenas um administrador pode criá-lo.

Os volumes são utilizados para armazenar buckets. Uma vez criado um volume, os utilizadores podem criar quantos buckets forem necessários. Ozone armazena os dados como chaves que residem dentro destes buckets.

O namespace do Ozone é composto por múltiplos volumes de armazenamento, que também são usados como base para a contabilização do armazenamento.

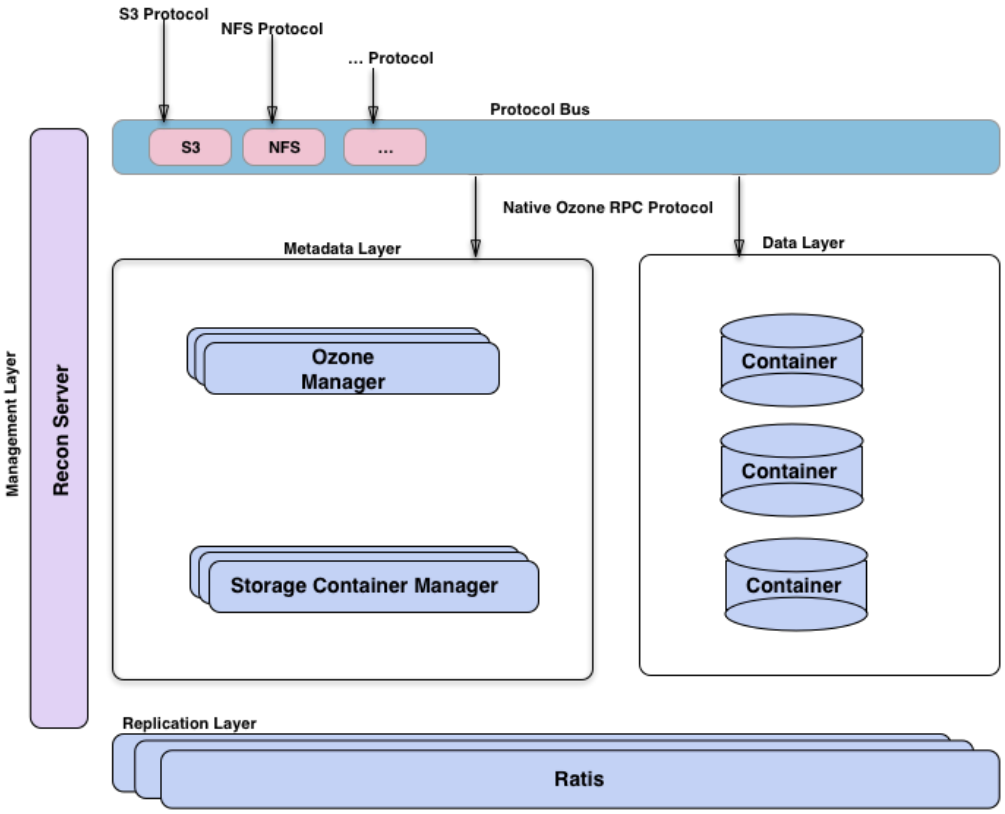
O diagrama de blocos abaixo ilustra os principais componentes do Ozone.



Armazenamento de Dados - Ozone

O Ozone Manager é responsável pela gestão do namespace, o Storage Container Manager gere a camada física e de dados, e o Recon funciona como a interface de gestão do Ozone.

Diferentes Perspetivas



Ozone Funcional

Qualquer sistema distribuído pode ser analisado sob diferentes perspetivas. Uma forma de ver o Ozone é imaginá-lo como o Ozone Manager, um serviço de namespace construído sobre o HDDS, um sistema de armazenamento de blocos distribuído.

Outra maneira de visualizar o Ozone é através das suas camadas funcionais. Existe uma camada de gestão de metadados, composta pelo Ozone Manager e pelo Storage Container Manager.

Existe também uma camada de armazenamento de dados, que é constituída pelos nós de dados e gerida pelo SCM.

A camada de replicação, fornecida pelo Apache Ratis, é usada para replicar metadados (OM e SCM) e garantir consistência quando os dados são modificados nos nós de dados.

Existe um servidor de gestão chamado Recon, que comunica com todos os outros componentes do Ozone e fornece uma API de gestão unificada e uma interface de utilizador para o Ozone.

Ozone possui um barramento de protocolo que permite a sua extensão através de outros protocolos. Atualmente, suporta apenas o protocolo S3, que foi implementado através do barramento de protocolo. Este barramento fornece um mecanismo genérico que possibilita a implementação de novos protocolos de sistema de ficheiros ou de armazenamento de objetos que interagem com o protocolo O3 Native.

Arquitetura do Apache Ozone

A arquitetura do Apache Ozone foi concebida para gerenciar milhares de milhões de objectos com escalabilidade e resiliência. Ela separa o gerenciamento de namespace do gerenciamento de blocos, usando os seguintes componentes principais:

Ozone Manager (OM):

- Gerencia o namespace (volumes, buckets e chaves).
- Utiliza o protocolo Ratis para garantir a consistência em clusters distribuídos.
- Storage Container Manager (SCM)**:
    - Gere os contentores, que são a unidade de replicação.
    - Coordena o posicionamento e a replicação de dados entre DataNodes.
- DataNodes:
    - Armazenam fisicamente os dados em contentores.
    - Garantem alta disponibilidade e replicação eficiente.
- Recon:
    - Ferramenta de monitorização e análise do cluster de Ozono.
    - Coleta métricas detalhadas para administração e diagnóstico.
- Protocolo Ratis**:
    - Implementa consenso distribuído para garantir replicação consistente.

Fluxo de operações no Ozono

- Registo de dados
    - O cliente solicita ao OM a alocação de blocos nos DataNodes.
    - Os dados são escritos diretamente nos DataNodes e replicados conforme necessário.
- Leitura de dados**:
    - O cliente pede ao OM para localizar dados.
    - Os dados são recuperados diretamente dos DataNodes.
- Replicação de dados**:
    - O SCM gere a replicação entre os DataNodes para garantir a tolerância a falhas.

Expectativas de desempenho

O Ozono emprega técnicas avançadas para atingir um elevado desempenho, tais como:

Replicação via RAFT para contentores abertos.

Replicação assíncrona para contentores fechados (dados frios).

Caraterísticas do Apache Ozone

O Apache Ozone oferece um conjunto robusto de recursos para atender às demandas de armazenamento massivo de dados. Aqui estão os destaques:

Compatibilidade S3:

APIs que permitem a integração com sistemas baseados em nuvem.

Ideal para migrações de dados e arquitecturas híbridas.

- Integração com o Hadoop**:

Alternativa direta ao HDFS, sem necessidade de adaptações a frameworks como o Apache Spark, Hive e YARN.

- Alta disponibilidade:

Baseado no Hadoop Distributed Data Store (HDDS), garante replicação consistente e tolerância a falhas.

- Suporte multimodal:

Permite o armazenamento como um sistema de ficheiros ou objetos, dependendo do caso de uso.

- Escalabilidade horizontal**:

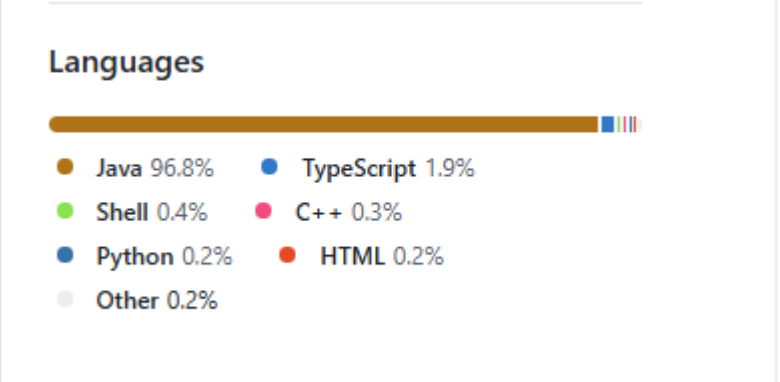
Suporta milhares de milhões de objectos, tornando-o ideal para aplicações de Big Data e Cloud Native.

- Reconhecimento de topologia**:

Otimiza os pipelines de leitura e gravação com base no posicionamento dos nós no cluster.

Detalhes do Projeto Apache Ozone

O Apache Ozone é construído principalmente em Java, que é a linguagem de base para todo o ecossistema Hadoop, incluindo HDFS, YARN e outros componentes relacionados. Essa escolha permite que o Ozone se integre perfeitamente ao Hadoop e às estruturas associadas, como Spark, Hive e MapReduce.



Detalhes do Projeto Apache Ozone

Fontes

Apache Software Foundation

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 aceder.

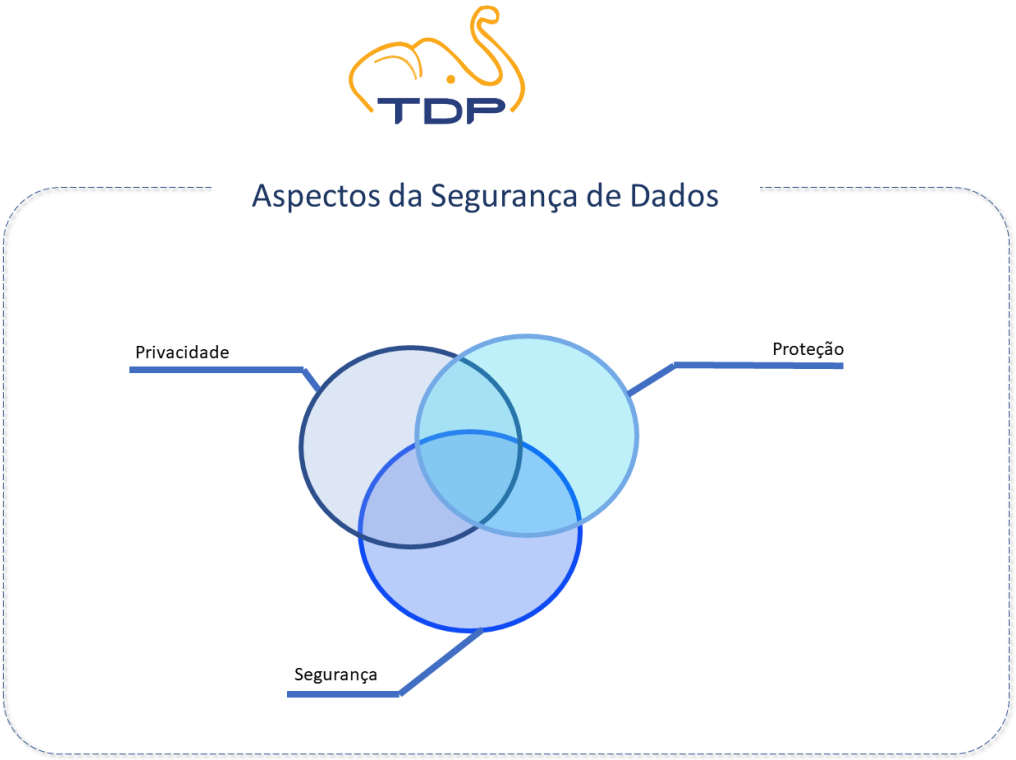
A segurança de dados refere-se às medidas de proteção empregadas para proteger informação digital contra acesso não autorizado, corrupção ou roubo, durante todo o seu ciclo de vida, preservando sua confidencialidade, integridade e disponibilidade.

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 monitorizar 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.



Aspectos da Segurança de Dados

Características do Apache Ranger

Apache Ranger é um framework criado para habilitar, monitorizar 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* num ambiente "multilocatário", o que, consequentemente, demandou uma estrutura de segurança de dados mais robusta, com suporte e monitorização 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 numa UI central ou usando REST APIs.
- **Autorização "detalhada":**
  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.
- **Aprimoramento do suporte a distintos métodos de autorização:**
  Controlo de acesso baseado em papéis, controlo 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 dum 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 controlo e visualizar atividades de auditoria.

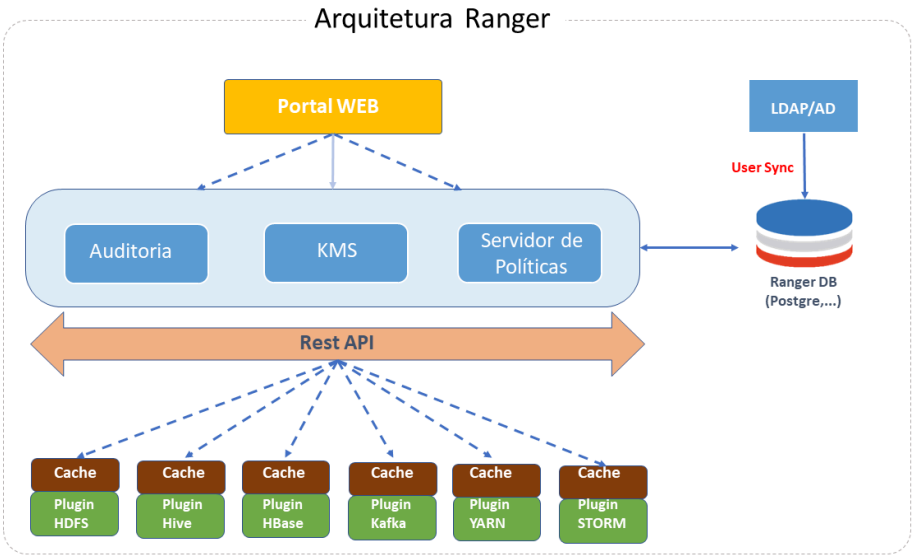
O portal, por sua vez, possui três partes distintas:

- **Auditoria:**
  Monitoriza 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 aceder 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 dum 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.



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

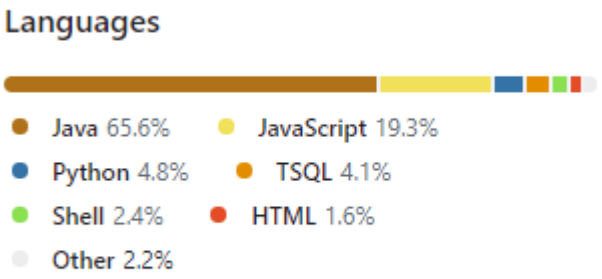
Componente	S/N
Ozone	SIM
Kudu	SIM
Kylin	SIM
NiFi	SIM
NiFi Registry	SIM
Sqoop *(obsoleto)*	SIM
Atlas	SIM
ElasticSearch	SIM
Presto	SIM
Schema Registry	SIM

Boas práticas para o Apache Ranger:

- Para autorização HDFS, mude o *umask* HDFS de 022 para 077 para prevenir que qualquer novo ficheiro 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 ficheiros 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 ficheiros e pastas deve ser criada se administradores desabilitarem esta política *default*.

Detalhes do Projeto Apache Ranger

Apache Ranger foi desenvolvido em Java predominantemente.



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 actual 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

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.

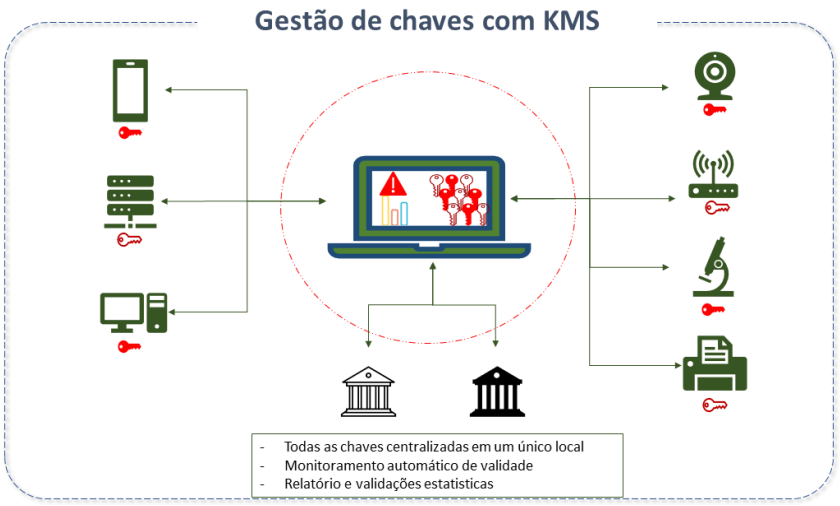
A ideia de gerenciar chaves criptográficas remonta aos primórdios da criptografia, mas o termo "KMS" tornou-se mais relevante com o aumento do uso de tecnologias de criptografia em ambientes empresariais.

Dentre suas características relevantes, destacamos:

- **Armazenamento seguro de chaves:** KMSs mantêm chaves criptográficas num 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;
- **Controlo de acesso:** Permitem um controlo rigoroso sobre quem pode aceder e usar as chaves;
- **Auditoria:** Registam 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 dum KMS varia, mas geralmente consiste num componente central de gerenciamento de chaves, um repositório seguro de chaves e mecanismos de controlo de acesso. HSMs são frequentemente usados para armazenar chaves de forma segura,

protegendo-as contra acesso não autorizado. As políticas de segurança são aplicadas para determinar quem pode aceder e usar as chaves.



Arquitetura básica dum 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 disponibilizadores 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 disponibiliza 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.

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

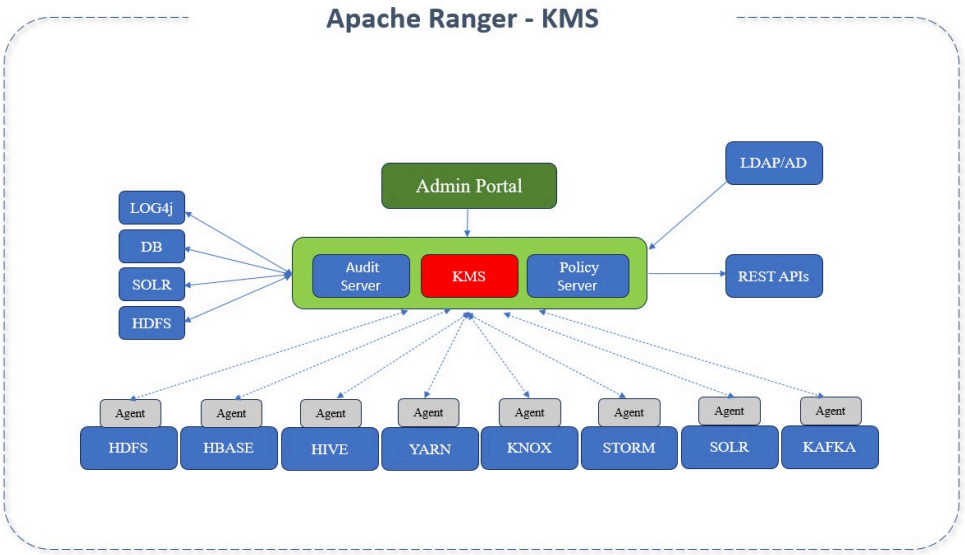
Dentre suas principais características, destacamos:

- **Gerenciamento seguro** de chaves criptográficas;
- **Integração com sistemas Hadoop** e outros sistemas de armazenamento;
- **Controlo de acesso** baseado em políticas para chaves;
- **Auditoria e registo 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:** Regista todas as atividades relacionadas a chaves para fins de auditoria;
- **Ranger Admin:** A interface de administração do Apache Ranger para configurar políticas de segurança.



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;
- **Controlo Fino:** Fornece um controlo mais detalhado sobre quem pode aceder e gerenciar chaves criptográficas;
- **Registo e Monitorização:** Facilita o registo e monitorização 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.
- **Controlo de Acesso Granular:** Para garantir que apenas as pessoas e os sistemas autorizados possam aceder as chaves criptográficas.

- **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 registar 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 equipas;
- **Monitorização Contínua:** Estabeleça um sistema de monitorização 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 usuários do Ranger KMS para garantir o uso correto e seguro do sistema;
- **Backup e Recuperação:** Implemente políticas para cópias de segurança 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.

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 operativos compatíveis com Java, tornando-o altamente portátil e escalável.

Fontes(s):

- Comunidade Apache

Apache Solr

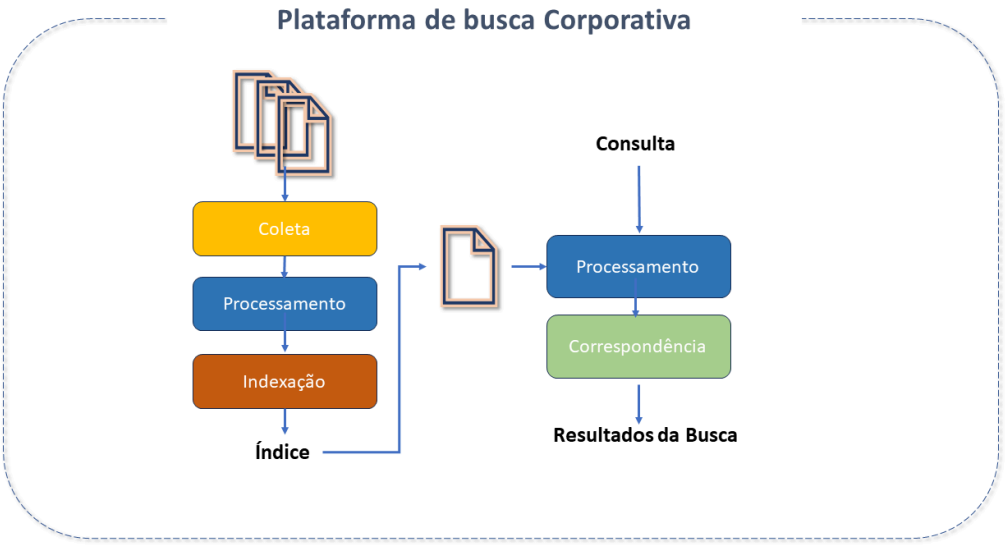
Plataforma de Pesquisa Corporativa

As Plataformas de Pesquisa Corporativa têm a função de criar uma funcionalidade de pesquisa segura, poderosa e fácil de usar.

Quando aplicadas em contextos empresariais, geralmente integram-se com soluções de inteligência de negócios e gestão de dados, sendo usadas para "limpar" e estruturar dados, facilitando a localização de informações. Podem extrair dados de diferentes fontes, como CRM, ERP, entre outros.

Para se qualificar como uma plataforma de pesquisa corporativa, o produto deve:

- Manipular informações de diferentes fontes, tipos de dados e formatos.
- Indexar ou arquivar dados.
- Oferecer opções de pesquisa inteligente.
- Disponibilizar uma interface para busca e recuperação de dados.
- Permitir que os utilizadores refinem as suas pesquisas com filtros avançados.
- Configurar permissões de acesso às informações.



Plataforma de Pesquisa Corporativa

Funcionalidades 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 capacidades de pesquisa ao site da empresa.

- Em janeiro de 2006, o seu código fonte foi doado para a Apache Software Foundation e, em 2007, já era considerado um projeto autónomo de nível superior (TLP), crescendo continuamente com recursos acumulados, atraindo utilizadores, colaboradores e *commiters*.
- Foi lançado pela Apache em 2008, na versão 1.3, e em 2010 foi fundido com o Lucene (o seu esquema de versionamento foi alterado em 2011 para coincidir com o do Lucene).
- Em 2020, a *Bloomberg* doou o operador Solr para o projeto Lucene/Solr, possibilitando a implementação e execução do Solr no Kubernetes.
- Em 2021, o Solr foi estabelecido como um projeto Apache independente (TLP), autónomo do Lucene. A sua primeira versão independente foi a 9.0.

O Solr é amplamente utilizado para casos de uso de pesquisa e análise corporativa. É desenvolvido de forma aberta e colaborativa pelo projeto Apache Solr na Apache Software Foundation. Tem uma comunidade de desenvolvimento muito ativa, com lançamentos regulares.

As suas principais funcionalidades incluem:

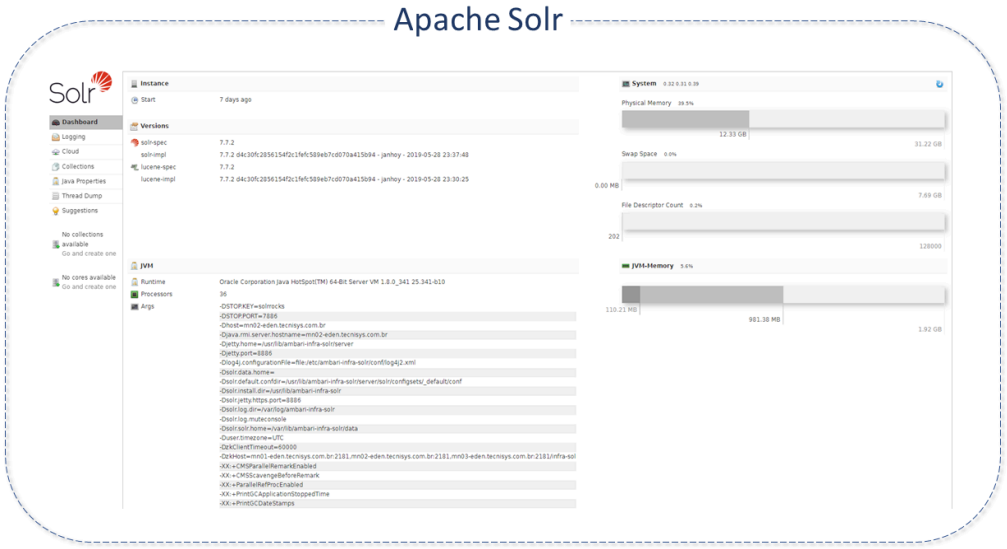
- **Funcionalidades Avançadas de Pesquisa em Texto:**
  Permite funcionalidades avançadas, como correspondência de frases, uso de *wildcards* para tornar as pesquisas mais flexíveis, além de suportar *joins* e agrupamentos em qualquer tipo de dado.

- **Otimizado para tráfego em grande escala**.
  Provado em escalas extremamente grandes ao redor do mundo.

- **Interfaces abertas:** Baseadas nos padrões XML, JSON e HTTP.

- **Interfaces administrativas abrangentes:** Inclui uma interface de administração responsiva integrada, facilitando o controlo das instâncias.

- **Monitorização fácil:** Publica cargas de dados de métricas via JMX.
- **Altamente escalável e tolerante a falhas:**
  - Construído sobre o Apache Zookeeper, facilita a escalabilidade.
  - Oferece replicação, distribuição, balanceamento e tolerância a falhas.
- **Flexível e adaptável com configuração simples:** Configurações de esquema flexíveis, permitindo armazenar quase qualquer tipo de dado no Solr.
- **Indexação em tempo quase real:** Permite que alterações sejam visíveis a qualquer momento.
- **Arquitetura extensível de plugins**. Publica vários pontos de extensão que facilitam a incorporação de plugins de indexação e consulta, permitindo a personalização do código à vontade (licença de código aberto).



Interface do Apache Solr

Arquitetura do Apache Solr

O Apache Solr foi projetado para escalabilidade e tolerância a falhas. Ele funciona como um servidor de pesquisa de texto independente.

O Apache Solr é executado sobre a biblioteca de software Lucene, que é, de fato, o motor que o alimenta. O núcleo do Solr é uma **instância do índice Lucene**, onde todos os ficheiros de configuração necessários para utilizá-lo estão localizados. A biblioteca de pesquisa Java Lucene é usada para indexação e pesquisa de texto completo e possui

APIs REST-like HTTP/XML e JSON, que tornam o Solr **utilizável nas linguagens de programação mais populares**.

A sua configuração externa permite adaptá-lo a muitos tipos de aplicações sem necessidade de codificação em Java, e a sua arquitetura de plugins suporta personalizações mais avançadas.

As consultas no SOLR são URLs de solicitações HTTP simples e as respostas são documentos estruturados em JSON, XML, CSV ou outro formato. Os documentos (*indexação*) são enviados para ele via JSON, XML, CSV ou binário através de HTTP. As consultas são realizadas via HTTP GET e recebem resultados em JSON, XML, CSV ou binário.

O Solr opera com uma **arquitetura não escrava**, pois **cada nó é o seu próprio mestre**. Os nós utilizam o 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 geridas nos ficheiros *schema.xml* e *solrconfig.xml*.

Os principais componentes da arquitetura do Apache Solr são:

- RequestHandler:
  As solicitações recebidas no Apache Solr são processadas pelo *Request Handler*. A solicitação pode ser uma consulta ou uma atualização de índice. Com base na solicitação, o *request handler* é selecionado. O handler geralmente é mapeado para um URI *end-point* através do qual a solicitação será atendida.

- Search Resource:
  Fornece a funcionalidade de pesquisa ao Apache Solr. Isso pode incluir *spell checking*, consultas, *faceting*, realce de ocorrências (*hit highlighting*), entre outros.

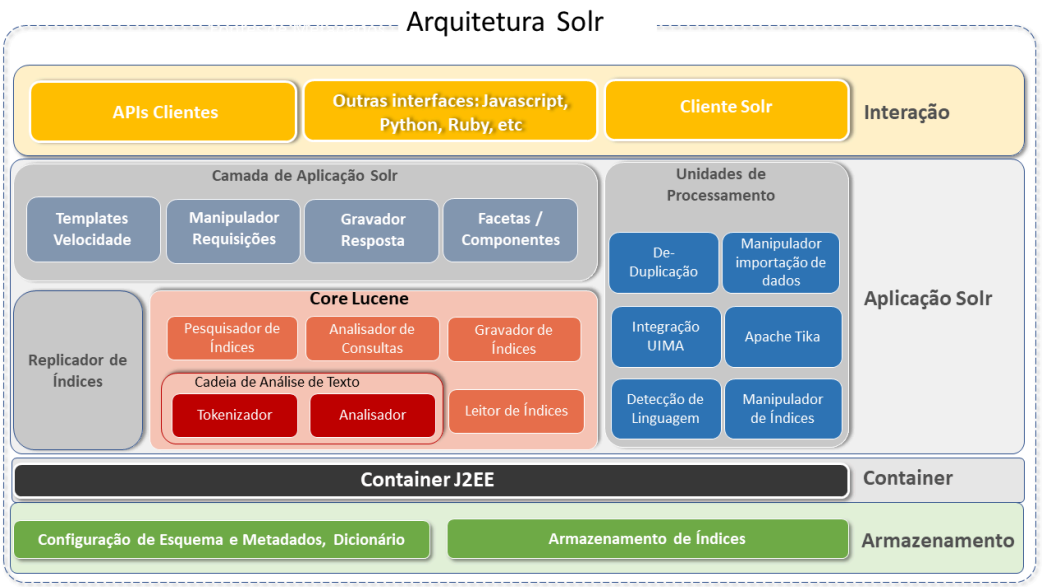
- Query Parsers:
  Analisam as consultas e verificam erros de sintaxe. Após a análise, traduzem as consultas para um formato compreensível pelo Lucene. O Solr suporta vários analisadores de consultas (*query parsers*).

- Response Writer:
  É o componente que gera a saída formatada para a consulta do utilizador. Os

formatos suportados incluem XML, JSON, CSV, entre outros. Existem diferentes *response writers* para cada tipo de resposta.

- Parser / tokenizer:
  O Lucene reconhece dados na forma de *tokens*. O Apache Solr analisa o conteúdo, divide-o em *tokens* e passa esses *tokens* para o Lucene. Um *Analyzer* no Apache Solr examina o texto dos campos e gera um fluxo de *tokens*. Um *tokenizer* divide o fluxo de *tokens* preparado para o parser em *tokens*.

- Update Request Processors:
  Cada solicitação de *update* é executada através dum conjunto de *plugins* (assinatura, registo, indexação), conhecidos coletivamente como "processador de solicitação de atualização" (*update request processor*). Este processador é responsável por modificações, como "remoção de campos, adição de campos, etc."



Arquitetura do Apache Solr

Termos do Apache Solr

- Cluster
  Nós Solr que operam em coordenação uns com os outros via Zookeeper e são geridos como uma unidade. Um *Cluster* pode conter várias coleções.

- Coleções:
  Um ou mais documentos agrupados num único índice lógico utilizando uma única configuração e esquema. Na nuvem, a coleção pode ser dividida em múltiplos fragmentos lógicos, que podem ser distribuídos por vários nós ou num único nó.

- Nó
  Na nuvem Solr, cada instância única do Solr é considerada um nó.

- Commit:
  Permite que as alterações ao índice sejam permanentes.

- Núcleo:
  Uma instância individual do Solr (representando um índice lógico). Vários núcleos podem ser executados num único nó.

- Shard (fragmento):
  É uma partição lógica da coleção que contém um subconjunto dos seus documentos, de forma que cada documento numa coleção esteja contido em exatamente um *Shard*.
  Quando um documento é enviado para um nó para indexação, o sistema determina a qual "shard" o documento pertence, em seguida, qual nó hospeda o seu *Líder*, encaminhando o documento para este líder para indexação, que por sua vez o encaminha para todas as outras réplicas.

- Replicação:
  Quando o índice é demasiado grande para uma única máquina e há um volume de consultas que *shards* individuais não conseguem suportar, é necessário replicar cada *shard*.
  No núcleo do Solr, a réplica é uma cópia do *shard* executada num nó.

- Líder:
  É a réplica do *shard* que distribui as solicitações do *Solr Cloud* para as outras réplicas.

Funcionalidades adicionais do Apache Solr

O Solr oferece capacidades ricas e flexíveis de pesquisa. As suas principais funcionalidades incluem:

Pesquisa em texto: Alimentado pelo Lucene, o Solr permite funcionalidades como correspondência de frases, *wildcards*, *joins*, agrupamentos e mais, para qualquer tipo de dados.

- Destaque (*Highlighting*):
  Permite que fragmentos de documentos correspondentes à consulta sejam incluídos na resposta.

- Faceting:
  Organização dos resultados da pesquisa em categorias com base nos termos indexados.

- Indexação em tempo real:
  O Solr permite criar um índice com vários campos ou tipos de entrada diferentes. Um índice Solr pode aceitar dados de várias fontes, como XML, CSV ou dados extraídos de tabelas em bases de dados e ficheiros em formatos comuns como Word ou PDF.

- Agrupamento e expansão de resultados:
  Agrupa documentos (colapsando o conjunto de resultados) de acordo com parâmetros e disponibiliza acesso ao documento no agrupamento "colapsado" para uso em exibição ou aplicação.

- Correção ortográfica (*SpellChecking*):
  Projetado para oferecer sugestões de consultas online com base em termos semelhantes.

- Integração com bases de dados.

- Funcionalidades NoSQL:**
  A funcionalidade *realtime get unique-key* permite a recuperação da versão mais recente de qualquer documento sem o custo de reabrir um pesquisador. Isso é especialmente útil ao usar o Solr como um armazenamento de dados NoSQL, não apenas como um índice de pesquisa.

  A funcionalidade *Real Time Get* depende do recurso de registo de atualizações (*update log*), que está ativado por padrão e pode ser configurado em `solrconfig.xml`.

Alguns dos sites mais famosos da Internet utilizam o Solr, como Macy, eBay e Zappos.

Melhores práticas para o Apache Solr

- **Campos indexados:**
  O uso de armazenamento em nuvem pode ser bastante otimizado ao controlar o número de campos indexados.
  Os índices representam um fator crítico entre qualidade, desempenho e custo.
  Muitos campos indexados podem degradar o desempenho e aumentar os custos.
  Ao definir campos, evite marcar campos como *indexed=true* se eles não forem utilizados numa consulta.
  O número de campos indexados aumenta:

  - O uso de memória durante a indexação.
  - O tempo de fusão de segmentos.
  - O tempo de otimização.
  - O tamanho do índice.
- **Armazenamento de campos:**
  Recuperar campos armazenados a partir do resultado de uma consulta pode ser custoso. Este custo é afetado pela quantidade de bytes armazenados por documento. Quanto maior a quantidade de bytes, mais esparsa será a distribuição dos documentos no disco - o que exige maior largura de banda de I/O para recuperar os campos desejados.

  Estes custos aumentam em implementações na nuvem. Obviamente, a sua utilização deve equilibrar qualidade, desempenho e custo.

- Uso da Cache do Solr:
  O Solr armazena em cache vários tipos de informações para evitar a repetição de consultas semelhantes.
  A Cache de Documentos ajuda a melhorar o tempo de resposta às solicitações.
  Existem três tipos principais de cache:

  - *Query cache*: Armazena os IDs dos documentos retornados por consultas.
  - *Filter cache*: Armazena filtros criados pelo Solr em resposta a filtros adicionados às consultas.
  - *Document cache*: Armazena os campos do documento solicitados ao exibir os resultados da consulta.
- Aquecimento automático (*Autowarming*):
  Ativar o *autowarming* pode aumentar significativamente o desempenho de uma pesquisa para qualquer um dos três tipos de cache.
  Ao aumentar *autowarmCount*, o Solr pré-preencherá ou *aquecera*

automaticamente o cache com os objetos criados pelo resultado da pesquisa.
Esta configuração indica o número de itens no *cache* que serão copiados para o novo "pesquisador".

- Expressões de Streaming (*Streaming Expressions*):
  O Solr oferece uma linguagem de processamento de *streaming* simples, mas poderosa, para o Solr Cloud.
  As expressões de *streaming* são um conjunto de funções que podem ser combinadas para realizar várias tarefas de computação paralela.

- Heap de memória da JVM (*JVM Memory Heap*):
  O *heap* é uma área de memória onde os objetos da aplicação, criados a partir de classes (com semântica de referência), são armazenados. Objetos que são referenciados em algum lugar são alocados lá.
  Esta área é gerida pelo *garbage collector* e ocupa espaço conforme necessário. No futuro, o GC liberará os espaços quando os dados numa porção não forem mais necessários.

  No caso da memória da JVM (Java Virtual Machine), grande parte da alocação já é reservada antecipadamente pelo GC, que tenta gerenciar a memória da melhor maneira possível.
  As configurações de *heap* da memória da JVM afetam diretamente o uso de recursos do sistema.
  Em um ambiente de nuvem, o uso de recursos e os custos podem aumentar rapidamente, afetando diretamente o desempenho e o sucesso ou fracasso de uma implementação de pesquisa.

  O tamanho ideal do heap de memória da JVM:
  Um intervalo entre 8 e 16 GB é adequado. Recomenda-se começar com o menor valor e realizar testes, monitorizando o uso de memória e ajustando gradualmente o tamanho até alcançar o ideal.

- Garbage Collector (*GC*):
  Este é o processo no qual a memória liberada num programa Java é devolvida ao pool de memória para reutilização.
  O coletor usado antes da versão 9 do Java era o ParallelGCf. O coletor G1GC opera de forma simultânea e multithreaded, resolvendo problemas de latência existentes no ParallelGC.
  O seu uso é, portanto, recomendado.

Além disso, o script de controlo do Solr vem com um conjunto de configurações pré-configuradas de GC Java, que funcionam bem para muitos tipos de cargas de trabalho. No entanto, essas configurações podem não ser adequadas para casos específicos.
Nesse caso, as configurações do GC podem ser alteradas usando a variável *GC_TUNE* no ficheiro `/etc/default/solr.in.sh`.

Recomendações de configuração para produção

- **Configurações de memória e GC:**
  Por padrão, o script `bin/solr` define o heap máximo como 512M. Para produção, ajuste esta configuração conforme necessário:

  ```
  solr_java_mem="-Xms10g -Xmx10g"
  ```

- **Encerramento devido à falta de memória:**
  O script `bin/solr` regista o script `bin/oom_solr.sh`, que é acionado em caso de erro *OutOfMemory*. Este script executa um comando `kill -9` no processo do Solr.

  Analise o conteúdo do ficheiro `/opt/solr/bin/oom_solr.sh` para entender as ações realizadas.

- **Modo SolrCloud:**
  Para executar o Solr no modo SolrCloud, configure a variável `ZK_HOST` para apontar para o conjunto Zookeeper. Por exemplo:

  ```
  zk_host=zk1,zk2,zk3
  ```

  Quando configurada, o Solr iniciará no modo de nuvem (*cloud mode*).

- **Zookeeper:**
  Para isolar os *znodes* do SolrCloud, use o suporte *chroot* do Zookeeper. Por exemplo:

  ```
  zk_host=zk1,zk2,zk3/solr
  ```

Antes de usar o *chroot* pela primeira vez, crie o caminho raiz no Zookeeper com o comando:

```
bin/solr zk mkroot /solr -z <ZK_NODE>:<ZK_PORT>
```

- **Definir o nome do host do Solr:**
  Configure o nome do host do servidor Solr utilizando a variável `SOLR_HOST`. Por exemplo:

  ```
  solr_host=solr1.example.com
  ```

  Isso é especialmente útil no modo SolrCloud, pois define o endereço do nó ao ser registado no Zookeeper.

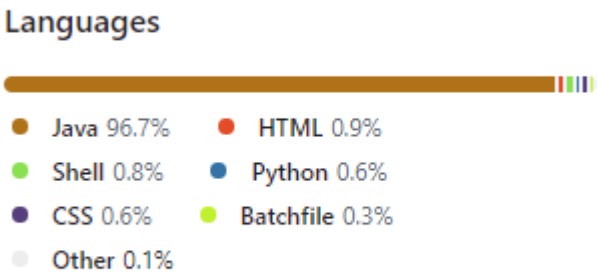
- **Substituição de configurações no solrconfig.xml:**
  Propriedades de configuração no ficheiro `solrconfig.xml` podem ser substituídas utilizando propriedades de sistema Java com a sintaxe `-Dproperty=value`.

- **Execução de múltiplos nós Solr por host:**
  O script `bin/solr` suporta a execução de várias instâncias numa única máquina. Contudo, essa configuração não é recomendada para instalações padrão, pois aumenta o consumo de recursos de CPU e memória.

  > **AVISO**
  >
  > Em cenários de escalabilidade extrema, a execução de vários nós Solr em um único host pode reduzir a necessidade de *heaps* muito grandes. Para mais detalhes, consulte a documentação.

Detalhes do Projeto Apache Solr

O Solr é escrito em Java.



Linguagem do Solr

Fontes

- whismy Apache Solr
- Apache Wiki
- Apache.Solr.org
- GitHub

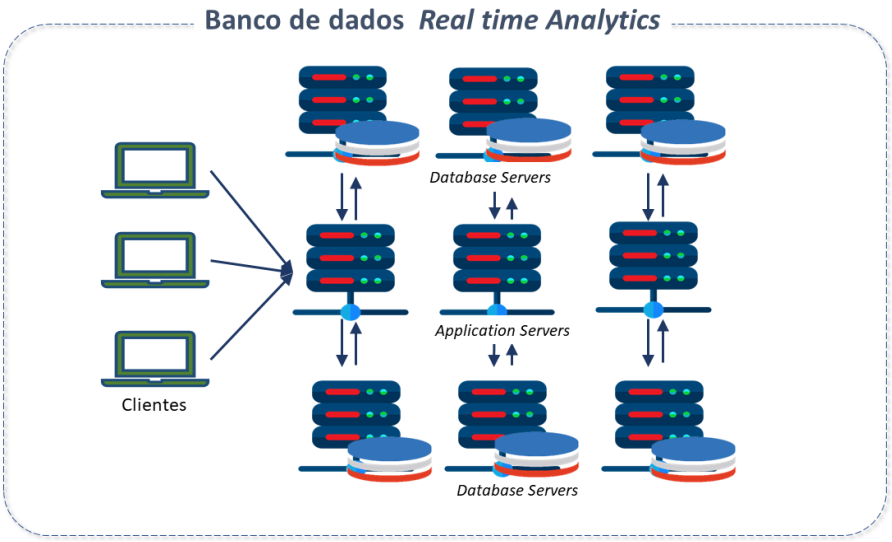
Apache Spark

Plataforma de Computação Distribuída

Com a evolução das redes de computadores, um novo paradigma computacional surgiu e tornou-se 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.

Este 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

Software Foundation em 2013.

Desenvolvedores de mais de 300 empresas ajudaram na sua implementação, e uma vasta comunidade com mais de 1.200 desenvolvedores de centenas de organizações continuam a contribuir para o seu contínuo detalhamento.

É usado por organizações em vários setores e a 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 planeador 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, treino de modelos para aprendizagem automática e IA.

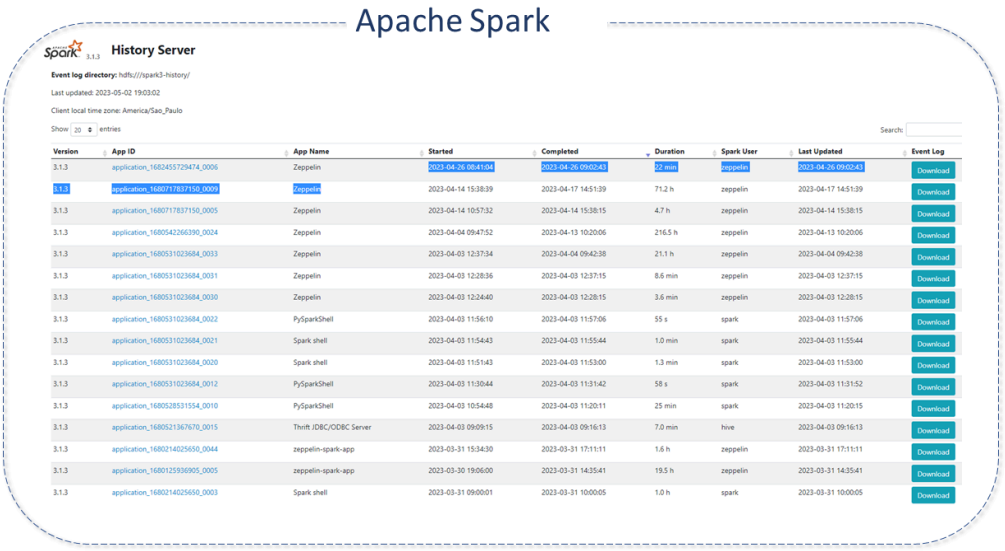
Adicionalmente, o Spark executa uma pilha de bibliotecas nativas de aprendizagem automática e processamento de grafos e estruturas de dados semelhantes a SQL, permitindo um desempenho excecional.

Conta com mais de 80 operadores de alto nível, facilitando a criação de aplicações paralelos.

O Spark partilha algumas semelhanças com o Hadoop. Ambos são estruturas de código aberto para processamento de dados analíticos, vivem na Apache Software Foundation, contêm bibliotecas de aprendizagem automática 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 as suas funcionalidades e disponibiliza ferramentas adicionais, como a implementação de aplicações num cluster Hadoop (com SIMR - Spark Inside MapReduce) ou YARN.



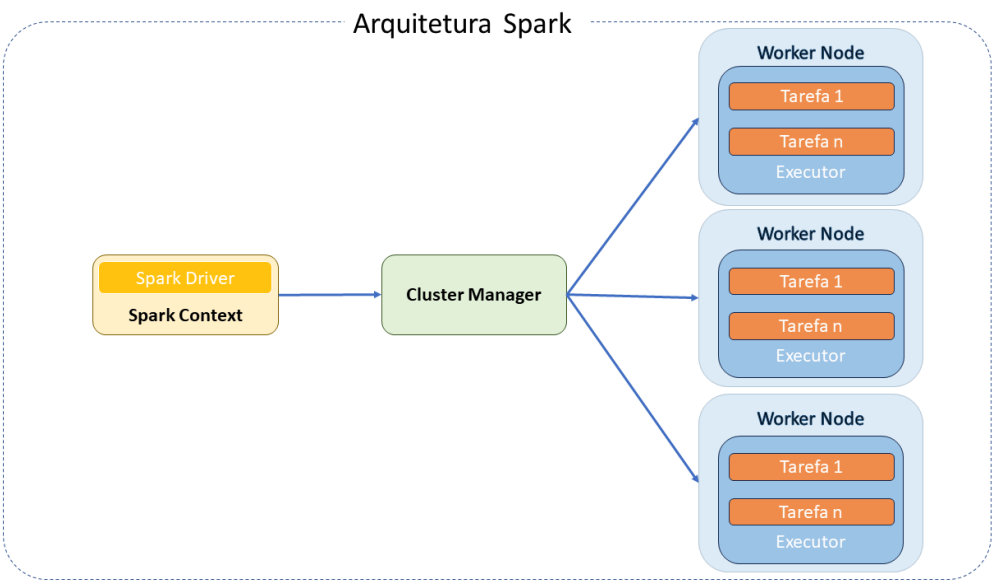
Interface Apache Spark

Arquitetura do Apache Spark

O Apache Spark é um mecanismo de processamento distribuído que opera no princípio de coordenador/trabalhador.

A sua arquitetura consiste nos seguintes componentes principais:

- **Spark Driver:** É o mestre da arquitetura Spark. É a aplicação principal que gere 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.



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 ficheiros tradicional, no HDFS e em algumas bases de dados NoSQL, como o HBase.

- **Operações**: Representam transformações (agrupamentos, filtros, mapeamento de dados) ou ações (contagens e persistências) realizadas num RDD. Um programa Spark normalmente é definido como uma sequência de transformações ou ações realizadas num conjunto de dados.

- **Spark Context**: O contexto é o objeto que conecta o Spark ao programa que está a ser desenvolvido. Pode ser acedido como uma variável num programa.

Bibliotecas do Apache Spark

Além das APIs, existem bibliotecas que compõem o seu ecossistema e disponibilizam capacidades adicionais:

- **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 dados em tempo real. É escalável, possui alto *throughput*, é tolerante a falhas e suporta workloads `em lote` ou `streaming`. O Spark Streaming permite a leitura/escrita 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. Isto 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 Bases de dados), transformá-los e expô-los para consultas *ad-hoc*.

- **Spark MLlib**: A biblioteca de aprendizagem automática 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 Partilhadas**: O Spark oferece dois tipos de variáveis partilhadas para torná-lo mais eficiente na execução em clusters:

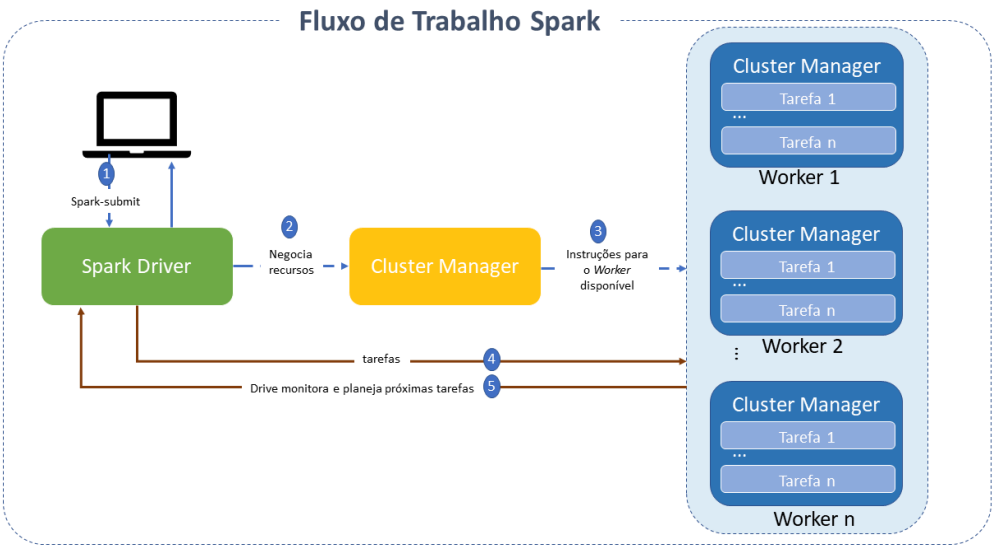
    - **Broadcast:** São variáveis apenas de leitura (*read-only*) que são armazenadas em cache em todos os nós do cluster para acesso ou uso por tarefas. Em vez de enviar os dados junto com cada tarefa, o Spark distribui as variáveis de transmissão (*broadcast*) na máquina usando algoritmos eficientes de transmissão para reduzir os custos de comunicação.

    - **Accumulator (Acumuladoras):** São variáveis partilhadas 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

O seu ciclo de vida envolve vários passos intermédios, cada um responsável por conduzir responsabilidades específicas.

- O processo começa 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 da aplicação.
- O programa driver solicita recursos do cluster manager para iniciar os executores com base na configuração da aplicação.
- O cluster manager ativa o executor no nó worker em nome do driver Spark, que agora assume a propriedade do ciclo de vida da aplicação.
- 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 aceder a **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**

    - **armazenamento em cache de dados in-memory**
    - **reutilização de datasets**
- Suporta **vários formatos e conjuntos de APIs** para lidar com qualquer tipo de dados em modo distribuído.
- Oferece um **mecanismo otimizado com suporte a grafos 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. Estas 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 ficheiro de saída, e outro job do Spark consome-o, faz alguns cálculos e grava novamente em algum ficheiro de saída. Nesse cenário, usar um formato de ficheiro serializado, como Parquet, dá-nos uma vantagem significativa sobre os formatos CSV e JSON.

- **Evitando Funções Definidas pelo Utilizador**: 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. Desta forma, perdemos todos os recursos de otimização oferecidos pelo Dataframe/Dataset do Spark.

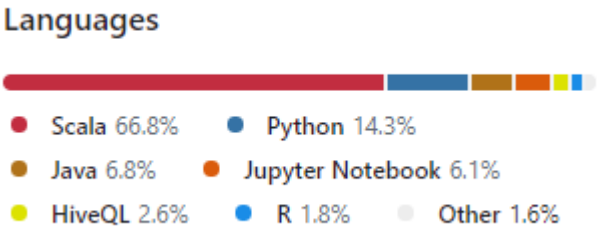
- **Dados de Memória em Cache**: Sempre que fazemos uma sequência de transformações do Dataframe e precisamos usar um Dataframe intermédio

repetidamente para cálculos adicionais, o Spark disponibiliza um recurso para armazenar um DF específico na memória na forma dum cache.

Detalhes do Projeto Apache Spark

O Spark é escrito na linguagem Scala e executado numa máquina virtual Java. Atualmente, suporta as seguintes linguagens para desenvolvimento de aplicações:

- 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 actual é **4.0.0**, distribuída pelo Helm Chart `tdp-spark` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- Apache.org
- GitHub

Apache Sqoop

Transferência de Dados

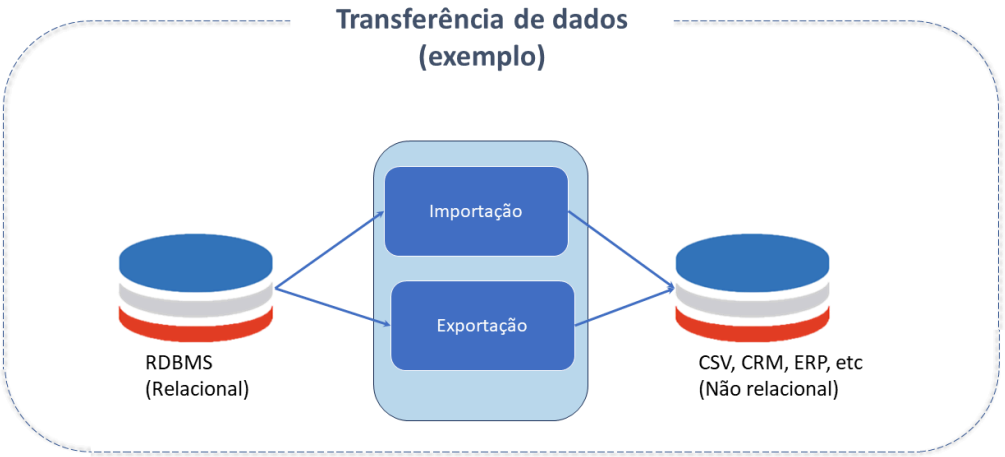
Este componente foi aposentado pela Apache Software Foundation e não recebe mais atualizações ou suporte.

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 dum sistema de "machine learning" pode incluir uma mistura de fontes e formatos.



Transferência de Dados

Características do Apache Sqoop

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 da Base 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 ficheiro 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 ficheiros**.
- **Paraleliza conexões em Base de dados** executando comando SQL como SELECT(import) e Insert/Update(export)
- O **Padrão do ficheiro importado do HDFS é o CSV**.
- **Conversão de tipos de dados**: O Sqoop **importa tabelas individuais ou *Databases* inteiros** para ficheiros em HDFS.

- Um **conector JDBC genérico** é fornecido para conexão com qualquer Base 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**.

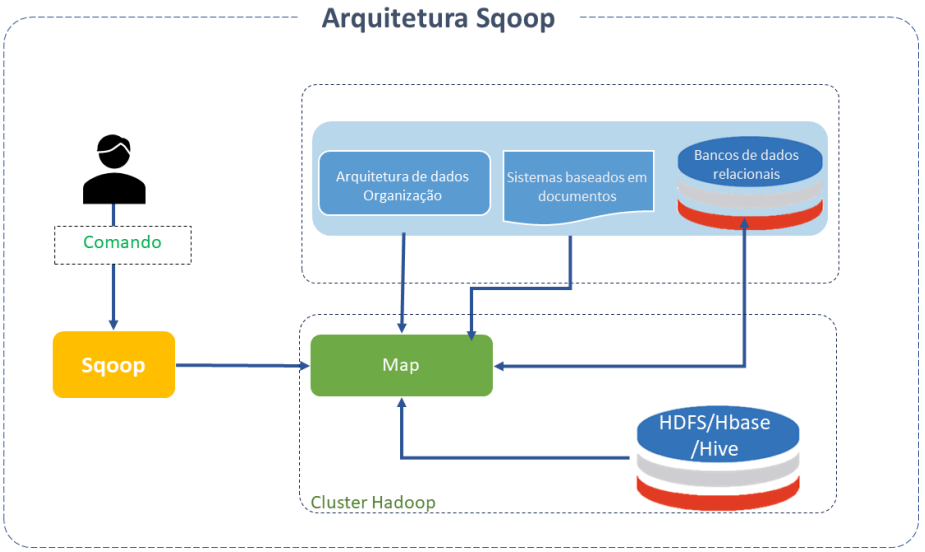
Arquitetura do Apache Sqoop

Sqoop disponibiliza 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 várias bases 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 da Base de Dados para deduzir os tipos de dados, manipulando cada registo de maneira segura.



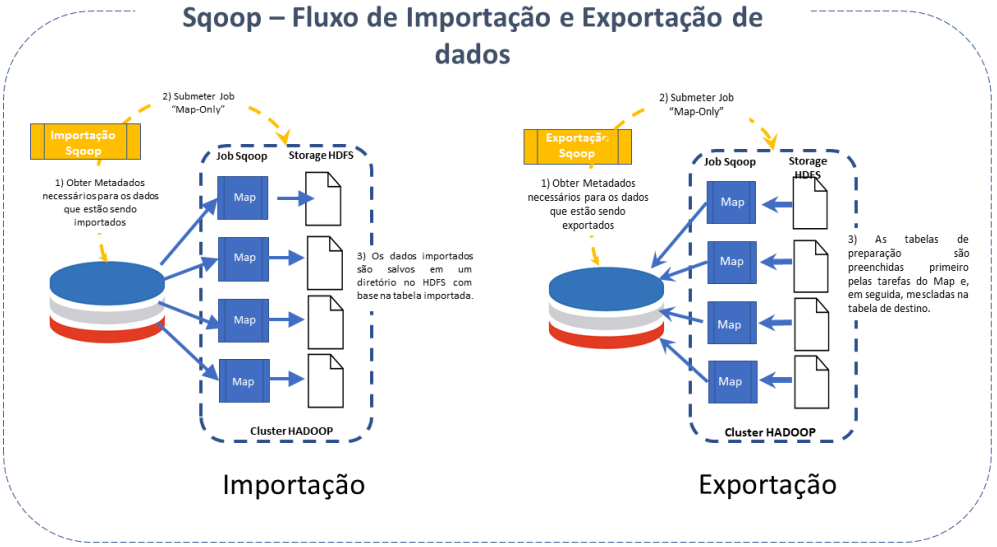
Arquitetura Sqoop

O Sqoop é composto, assim, de duas operações principais:

- Importação Sqoop

Procedimento realizado com o comando *Sqoop import*. Cada registo carregado no SGBD Hadoop como um único registo é mantido em ficheiros 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 ficheiros Hadoop para o SGBD Relacional. Antes de finalizar a operação, os dados exportados são transformados em registos.



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

- 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 a Base de Dados JDBC.

Boas Práticas para Apache Sqoop

- **Importação para formato binário** Embora as importações para o formato de ficheiros CSV sejam fáceis de testar, alguns problemas podem surgir quando o texto armazenado na Base 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 algumas bases de dados. Usando *-m* ou *--num--mappers* pode-se definir o grau de paralelismo no Sqoop.

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 dum 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) numa tabela e a usará como coluna de divisão.

  Os valores baixo e alto para a coluna de divisão são recuperados da Base 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
```

> **AVISO**
>
> O número de tarefas map deve ser menor que o número máximo de conexões de databases paralelos possíveis. O incremento no grau de

> paralelismo deve ser menor que aquele disponível dentro de seu Cluster Map Reduce.

- **Controlo do Processo de transferência de dados**: Um método popular de melhorar o desempenho 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 controlo de importação/exportação, na Guia do Usuário:

  - **Batch:** (em lote) Que significa que as instruções SQL podem ser agrupadas num lote quando os dados são exportados.

    > **NOTA**
    >
    > A interface JDBC disponibiliza uma API para fazer lotes numa 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 registos 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 registos que o Sqoop pode importar por vez.

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 disponibiliza um suporte razoável. No entanto, algumas bases de dados podem obter maior desempenho usando utilitários específicos de bases de dados pois são otimizados para disponibilizar a melhor velocidade de transferência possível, colocando menos pressão sobre o servidor da Base de Dados .

Ao disponibilizar 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
```

> **AVISO**
>
> Existem várias limitações que companham essa importação mais rápida. Nem todas as bases de dados possuem utilitários nativos

> disponiveis e esse modo não está disponível para todas as bases 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*.

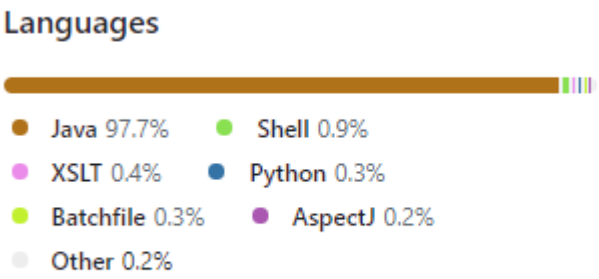
Idealmente, configuramos o parâmetro de consulta de limite como min(id) e max(id) juntamente com o nome da tabela.

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

Notas Técnicas

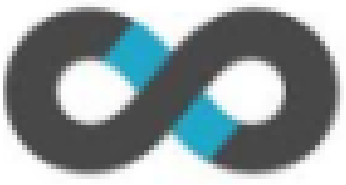
> **COMPONENTE APOSENTADO — REMOÇÃO PREVISTA NO TDP 3.1.0**
>
> O Apache Sqoop foi **aposentado pela Apache Software Foundation** e movido para o Apache Attic, repositório oficial de projetos inativos. Isto significa que o projecto já não recebe novas versões, correcções de segurança ou suporte da comunidade.
>
> A última versão estável disponível é a **1.4.7** (lançada em 2017).
>
> **Impacto para utilizadores do TDP:**
>
> - O Sqoop mantém-se funcional nas versões actuais da plataforma, mas não é recomendado para novos projectos.
> - **A partir do TDP 3.1.0, este componente será removido da plataforma.**

Fontes:

- Apache Sqoop
- Apache Attic — Sqoop

Apache Superset

Visualização



Visualização de dados consiste na representação gráfica de informações e dados. Elementos visuais como grafos 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 uma aplicação 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

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:

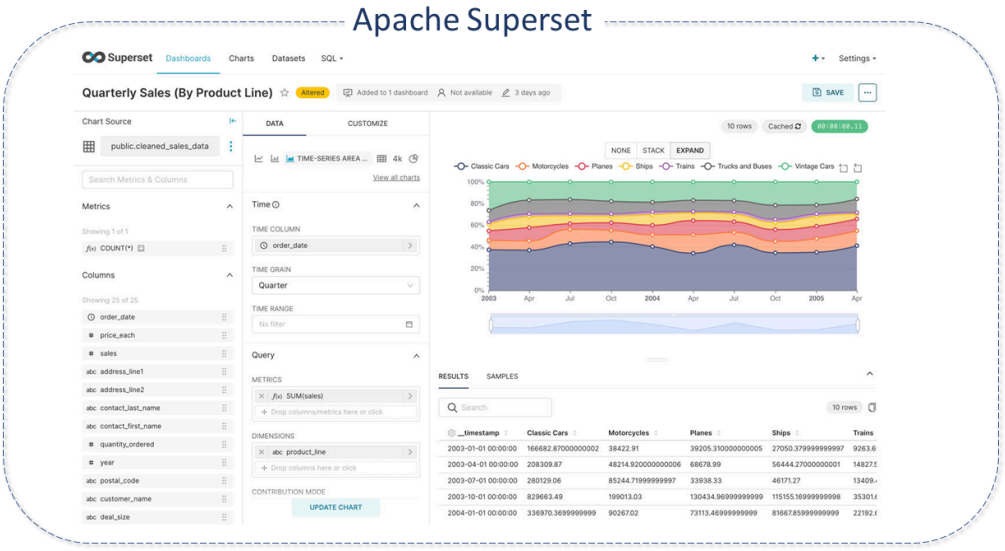
- **Interface Intuitiva**: Permite visualizar conjuntos de dados e criar painéis interativos.
- **SQL IDE**: Facilita o preparo de dados para visualização, incluindo um rico navegador de metadados.
- **Segurança**: É uma das suas principais vantagens, pois oferece controlo total sobre o acesso a dados. Permite a adição de usuários à Base 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 Bases de Dados SQL**: Compatível com a maioria das bases de dados que utilizam SQL.
- **Cache e Consultas Assíncronas**: Melhora o desempenho ao reduzir a carga de consulta diretamente nas bases de dados.
- **Extensibilidade**: Com controlo total de acesso ao dado, permitindo a configuração de regras complexas sobre quem pode aceder 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.

    - 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 aplicação:** Que operam de modo independente.



Interface do Apache Superset

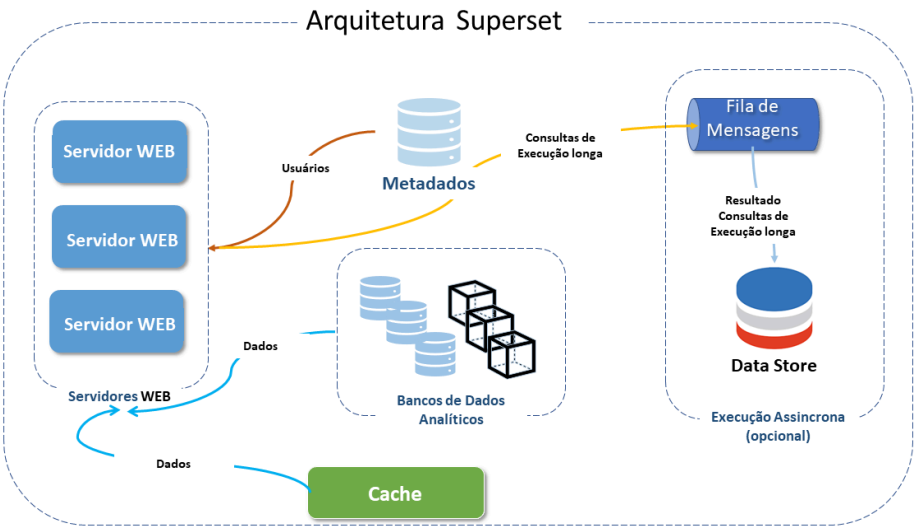
Arquitetura do Apache Superset

O Apache Superset opera com uma arquitetura centrada 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 esrutura 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 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 base 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 Base 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 grafos e dados. Cada seção dentro do *Dashboard* é chamada *Slice*, que, por sua vez, podem estar em diversos formatos: texto, grafo, 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 grafos, por exemplo.



Arquitetura do Apache Superset

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 Base 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 grafos 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 todas as principais bases 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

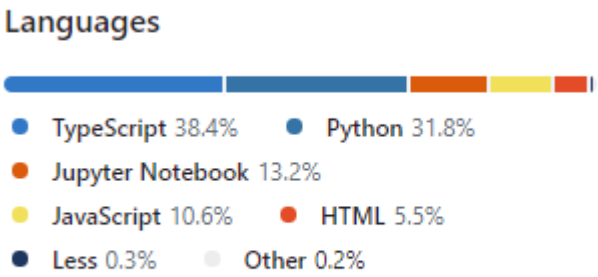
O Apache Superset disponibiliza uma ampla varidade de grafos, tabelas e layouts. Alguns exemplos são:

- Grafo de Dispersão.
- *Grids*.
- Polígonos.
- *Path*.
- *Screen Grids*.
- *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 num ambiente virtual. O *pyenv-virtualenv* pode ser instalado se o *pyenv* estiver a ser 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 actual é **5.0.0**, distribuída pelo Helm Chart `tdp-superset` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes

- Comunidade Apache Superset
- GitHub Apache Superset

Trino

Motor de Consulta Distribuído

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

Um mecanismo de consulta distribuído é concebido para executar consultas SQL complexas em grandes volumes de dados dispersos por várias fontes. Elimina a necessidade de mover ou duplicar dados, permitindo análises diretamente no local onde os dados estão armazenados, aumentando a eficiência e a escalabilidade.

Principais Características dum Mecanismo de Consulta Distribuído:

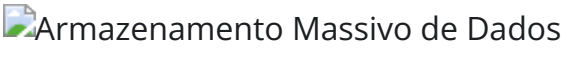
- **Execução Paralela:** Divide consultas em tarefas mais pequenas, que são processadas simultaneamente em diferentes nós dum cluster, otimizando o tempo de execução.
- **Ligação a Múltiplas Fontes de Dados:** Oferece integração com data lakes, data warehouses e bases de dados relacionais, permitindo a execução de consultas federadas.
- **SQL como Linguagem Universal:** Os analistas podem realizar consultas utilizando SQL, sem necessidade de aprender linguagens específicas.
- **Desempenho Otimizado:** Utiliza técnicas como *pushdown* de predicados e armazenamento em memória para reduzir a latência e maximizar a performance.

Entre os casos de uso mais comuns dos 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 aos *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.



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 responder às suas necessidades internas de análise de dados, o Trino surgiu em 2020 após uma divisão com a Presto Foundation. Desde então, tornou-se uma 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 aplicações 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:** Possibilita a execução de consultas em múltiplas fontes de dados, como HDFS, S3, bases de dados relacionais e data warehouses.
- **Alta Performance:** A 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.

- **Comunidade Aberta:** Desenvolvido sob a liderança da Trino Software Foundation, uma organização sem fins lucrativos.

Arquitetura do Trino

O Trino é um mecanismo de consulta distribuído que processa dados em paralelo em vários servidores. Os servidores dum Cluster Trino são classificados como *Coordinators* e *Workers*.

As secções seguintes 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 utilizadores ligam-se ao *Coordinator* através de ferramentas de consulta SQL. O *Coordinator* orquestra tarefas entre os *Workers* e acede às 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 (*Node*)

Um *Node* no Trino refere-se a qualquer servidor dentro dum cluster que executa um processo Trino. Normalmente corresponde a um único computador, uma vez que apenas um processo Trino é recomendado por máquina.

Coordinator

O *Coordinator* é o servidor central responsável por analisar instruções, planear consultas e gerenciar nós *Workers*. Atuando como o "cérebro" do cluster, rastreia as atividades dos *Workers*, coordena a execução de consultas e comunica-se 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* obtêm dados de conectores, trocam dados intermediários e comunicam-se com o *Coordinator* via API REST. Quando iniciado, um *Worker* regista-se no servidor de descoberta do *Coordinator* para alocação de tarefas.

Cliente

Os clientes ligam-se ao Trino para enviar consultas SQL e recuperar resultados. Podem aceder às fontes de dados configuradas por meio de catálogos e incluem ferramentas como interfaces de linha de comandos, aplicações de ambiente de trabalho 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*, bases de dados relacionais e armazenamentos de chave-valor. O acesso a estas fontes de dados é configurado através de catálogos, que definem os conectores necessários, credenciais e outros parâmetros.

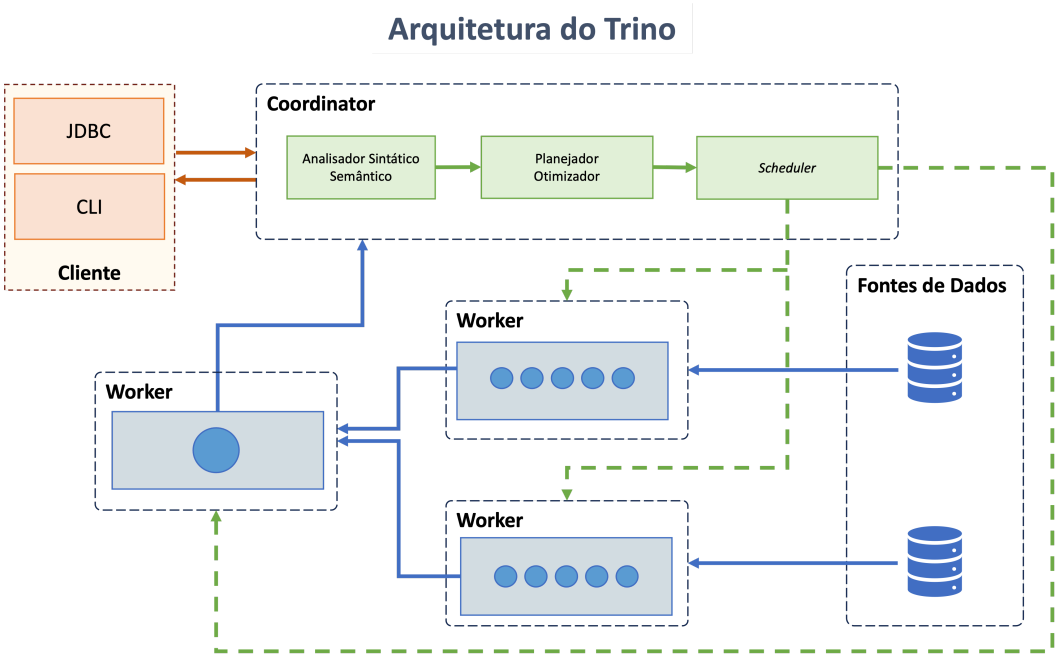
A seguir apresentamos os conceitos fundamentais associados às Fontes de Dados no Trino:

- **Conector:** Os conectores permitem que o Trino interaja com fontes de dados específicas, funcionando como *drivers* de bases 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 aceder a uma fonte de dados. Os catálogos são definidos em ficheiros de propriedades armazenados no diretório de configuração do Trino. Um catálogo pode conter esquemas e tabelas, permitindo o acesso a múltiplas fontes de dados num único cluster.

- **Esquema:** Os esquemas organizam tabelas e outros objetos dentro dum catálogo. Correspondem a conceitos semelhantes em bases de dados como Hive e MySQL.

- **Tabela:** Uma tabela consiste em linhas desordenadas organizadas em colunas nomeadas com tipos específicos. As tabelas são acedidas por nomes totalmente qualificados enraizados em catálogos.



Arquitetura do Trino

Modelo de Execução de Consultas do Trino

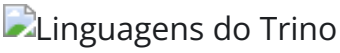
O Trino executa instruções SQL transformando-as em consultas distribuídas executadas no cluster.

Seguem mais detalhes sobre os termos utilizados no modelo de execução de consultas do Trino:

- **Instrução (*Statement*)**: O texto SQL enviado ao Trino, convertido num plano de consulta durante a execução.
- **Consulta (*Query*)**: Engloba os componentes e a configuração necessários para executar uma instrução, incluindo estágios, tarefas, divisões e operadores.
- **Estágio (*Stage*)**: Secções dum plano de consulta distribuído, organizadas hierarquicamente.
- **Tarefa (*Task*)**: Executa estágios em paralelo nos *Workers*.
- **Split (*Divisão*)**: Um segmento dum conjunto de dados maior, processado por uma tarefa.
- **Driver**: A menor unidade de paralelismo, combinando operadores para processar dados dentro de uma tarefa.
- **Operador**: Realiza transformações nos dados, como buscas em tabelas ou filtros.

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 actual é **478**, distribuída pelo Helm Chart `tdp-trino` v3.0.1.
>
> Para detalhes de configuração, consulte a documentação no TDP Kubernetes.

Fontes: Trino - Overview

Apache Zeppelin

Notebook

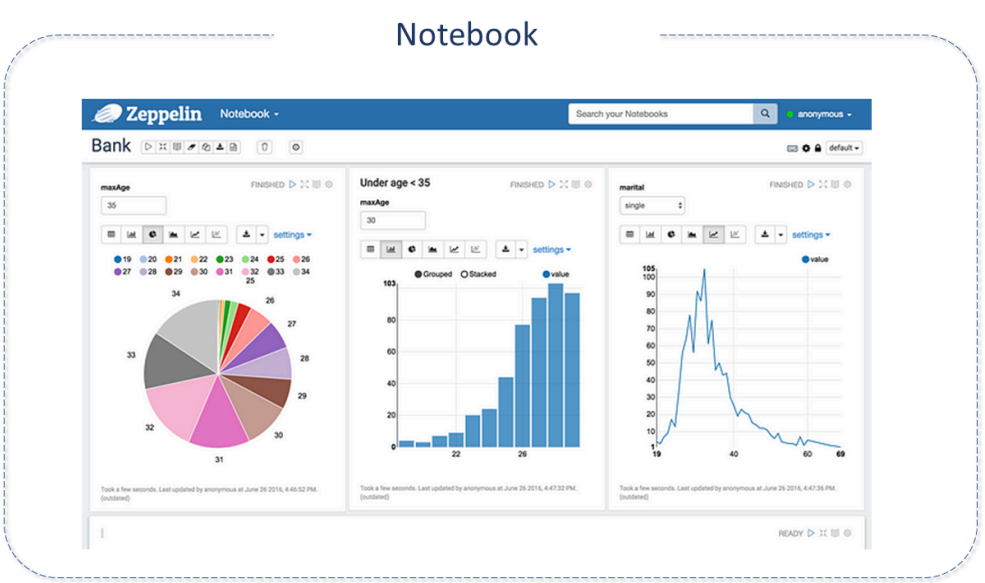
Tipicamente utilizado por cientistas de dados em experimentos e tarefas exploratórias, o notebook é uma ferramenta de computação interativa que permite aos usuários escrever e executar código, visualizar resultados e partilhar insights.

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 disponibiliza recursos de ingestão, exploração, visualização, partilhamento 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".



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.

- **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 disponibiliza:

    - SparkContext e SQLContext automáticos.
    - Carga de "dependência" JAR em tempo de execução a partir do sistema de ficheiros local ou repositório "Maven".
    - Cancelamento de trabalho com exibição do progresso.
- **Visualização de Dados**: Alguns grafos básicos também podem ser utilizados.

    As visualizações não se limitam a query SPARKSQL.

    Qualquer saída de qualquer linguagem "backend" pode ser reconhecida e visualizada.

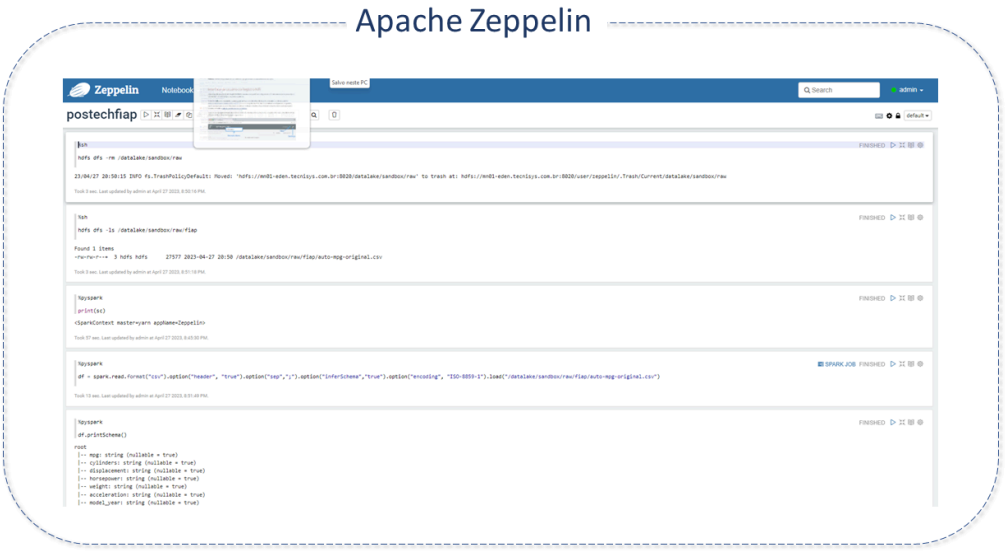
    - **Grafos dinâmicos**: Apache Zeppelin agrega valores e os mostra em grafos dinâmicos. Com um simples drag-and-drop, é possível criar um grafo com múltiplos valores agregados, incluindo soma, contagem, média, mínimos e máximos.

    - **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 partilhada e, em seguida, o Zeppelin pode transmitir todas as alterações em tempo real, bem como a colaboração no Google Docs.

    Pode, ainda, disponibilizar 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.



Interface Zeppelin

Arquitetura do Apache Zeppelin

O Apache Zeppelin é dividido em 03 camadas:

- **Front-end**: A partir dum navegador Web, o utilizador tem contacto 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**:

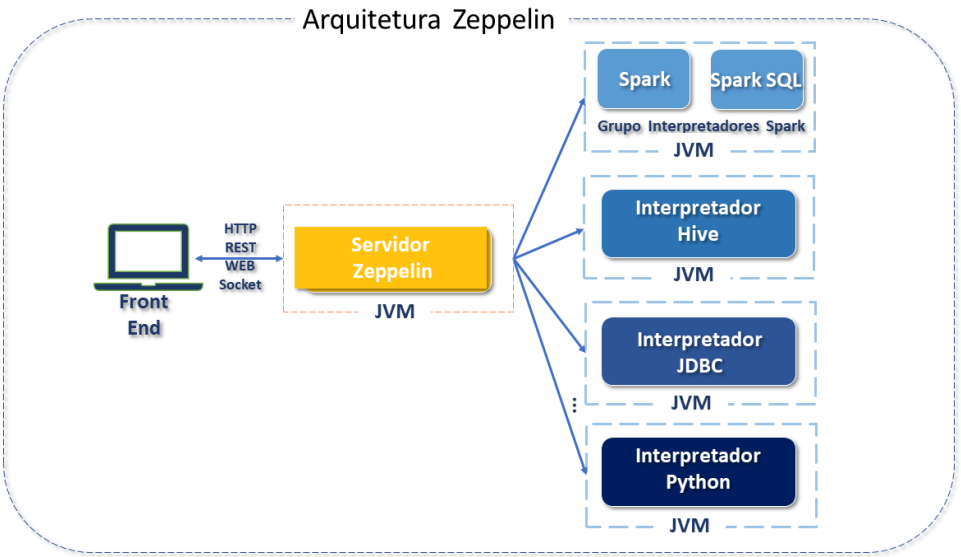
  - **REST**: Estilo de arquitetura para definir **restrições e propriedades do protocolo *HTTP***.
  - **Web Socket**: Tecnologia para **comunicação bidirecional por canais *full-duplex*** - transmissor e receptor podem transmitir dados simultaneamente em ambos os sentidos sobre um único socket TCP.
- **Zeppelin Server**: O servidor **opera numa 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** num ficheiro de definição simples.

Tomando esse ficheiro 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 "conectável" 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**.

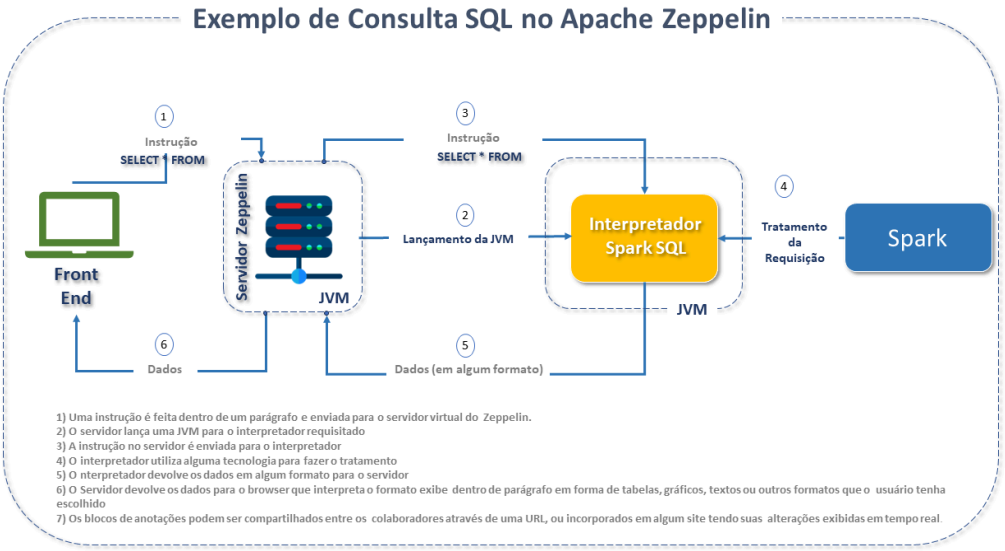
Cada parágrafo é uma caixa que **recebe algum tipo de script pré-definido nos interpretadores**.

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.

A interação entre o utilizador, 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.

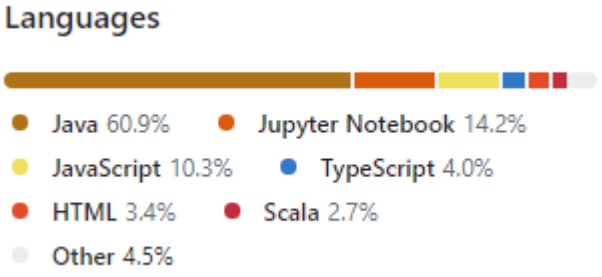
- **Segurança**: Como qualquer software, **a segurança depende da matriz de ameaças e das opções de implantação**:

    - Autenticação
    - Kerberizar o Cluster usando Ambari

    - 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 disponibiliza propagação de identidade ideal.
    - **Escolhendo os interpretadores**: Por default, Zeppelin registará 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

Fontes:

- Apache Zeppelin

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.

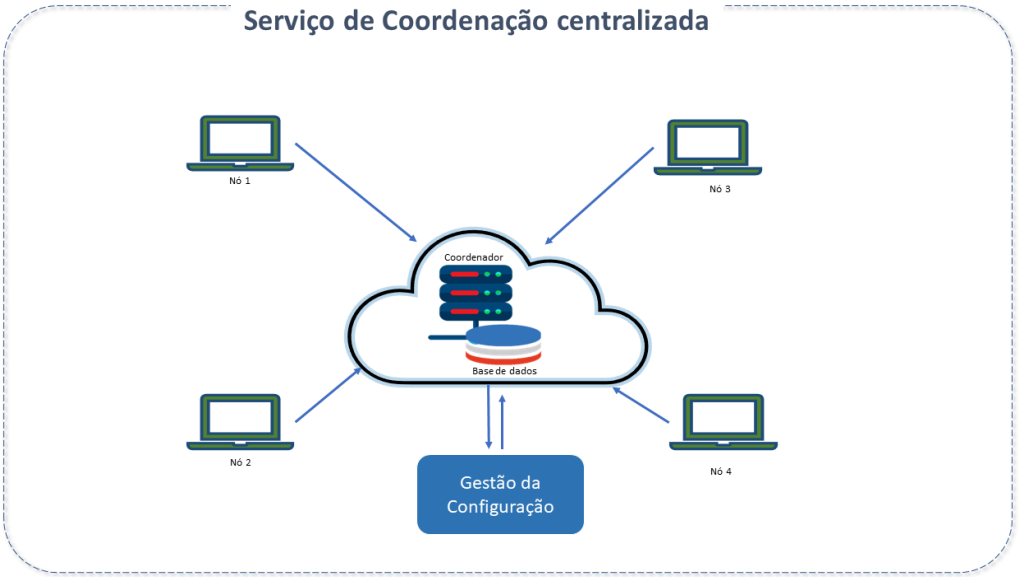
Obviamente, este sistema está sujeito a falhas de hardware, travamentos, falhas de sistema, de comunicação e assim por diante, São falhas que não seguem um padrão e, portanto, tornam difícil aplicar um código de tolerância na lógica do aplicativo ou no projeto do Sistema.

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 do desempenho 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 disponibilizam um plano e uma versão única e o estado pretendido dos sistemas da organização, além de dar visibilidade de quaisquer modificações na configuração, por meio de trilhas de auditoria e rastreamento de alterações.



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 dum 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 num _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

a correta coordenação:

- Ao contrário dos sistemas de ficheiros convencionais, o Zookeeper disponibiliza **alto rendimento e baixa latência**.
- Executa numa 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 registem para receber notificações sempre que um *znode* sofre alguma alteração (criação ou exclusão, mudanças nos dados mantidos pelo *znode*, criação ou exclusão dum dos filhos na sub-árvore dum _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 num 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 registadores de dados, denominados *znodes* (cada nó da árvore), o que o torna uma solução eficiente para gerenciamento de informações de configuração.

Os **clientes** são nós que **consomem** os serviços e os **servidores** são nós que *disponibilizam* os serviços.

Os **caminhos** para os nós são expressos como caminhos **canônicos** (de acesso mais curto a um ficheiro a partir do diretório raiz), absolutos e separados por barra.

Não há referência relativa.

A principal diferença entre Zookeeper e um sistema de ficheiros padrão é que **todo *znode* pode ter dados associados como filhos** (todo ficheiro 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 dum 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 Base de Dados do servidor.

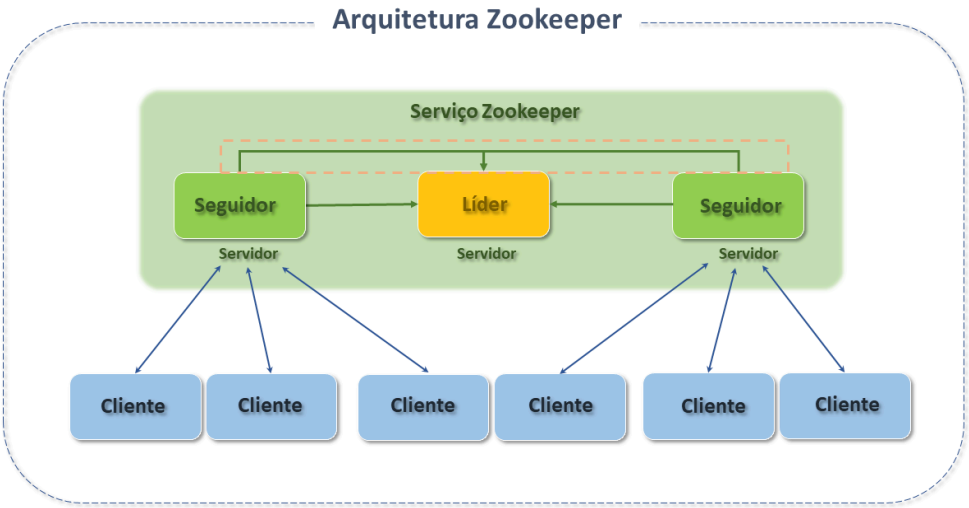
Se a solicitação registar uma observação num *znode*, esta também será rastreada localmente no servidor.

As solicitações de escrita, 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 escrita 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.

Zookeeper segue um **modelo de armazenamento partilhado e hierárquico** similar a um sistema de ficheiros 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) num 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 da Base 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 registos de dados conhecidos como znodes, similares a ficheiros 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 dum *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 dum 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

os clientes, dependendo da política de **Lista de controlo 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.

> **NOTA**
>
> É atribuído um número de sequência pelo Zookeeper como parte de seu nome durante sua criação. O valor dum 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 ficheiro 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* num dispositivo ocupado, pois isto prejudicará o desempenho. No caso de existir apenas um dispositivo de armazenamento,

sugerimos colocar os ficheiros 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*.
    - A Monitorização de sistema como *vmstat* pode ser usado para monitorizar estatísticas de armazenamento virtual e decidir sobre o tamanho ideal de memória dependendo das necessidades do aplicativo. Em qualquer caso, sempre evitar o *swap*.

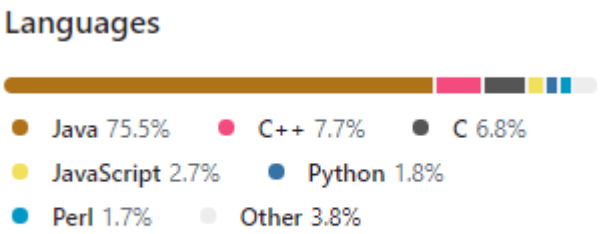
Coisas a fazer:

- 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 ficheiros de *log* transacionais.
- Além disso, é importante avaliar a necessidade de cópias de segurança, que, sendo o Zookeeper um serviço replicado, só é necessário num dos servidores do conjunto.
- Zookeeper usa o Apache *log4j* como infraestrutura de registo. É recomendável definir o *rollover* automático usando o recurso *Log4j* embutido para *logs* do Zookeeper, à medida que os ficheiros 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.



Linguagens do Zookeeper

Fontes:

- Wiki.Apache
- Zookeeper.apache.org
- Zookeeper GitHub

PARTE II - INÍCIO RÁPIDO

Sandbox TDP

Error loading this section: GET http://127.0.0.1:3002/tdp/pt-PT/commons/sandbox -> 404

PARTE III - INSTALAÇÃO DOS COMPONENTES

Minimum Requirements

Os requisitos mínimos para a implementação dum *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 cumprir 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 controlo, 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*)

> **NOTA**
>
> Clusters com um número reduzido de máquinas podem ser implementados, desde que se respeitem os requisitos técnicos, o dimensionamento e a 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.

> **AVISO**
>
> 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.

Caso necessite de apoio, entre em contacto através da Área de Suporte.

> **NOTA**
>
> 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 Operativos Suportados

Atualmente, o TDP está disponível para os seguintes sistemas operativos:

- CentOS 7.5 e versões minoritárias superiores;
- CentOS 8.x e 9.x;
- Red Hat Enterprise Linux 7.5 e versões minoritárias superiores;
- 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 aceder as interfaces gráficas dos componentes dum *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.

Support Tools

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 disponibiliza diversos software complementares para distribuições Enterprise Linux, como ferramentas de administração, desenvolvimento, monitorização, 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 ficheiros 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
```

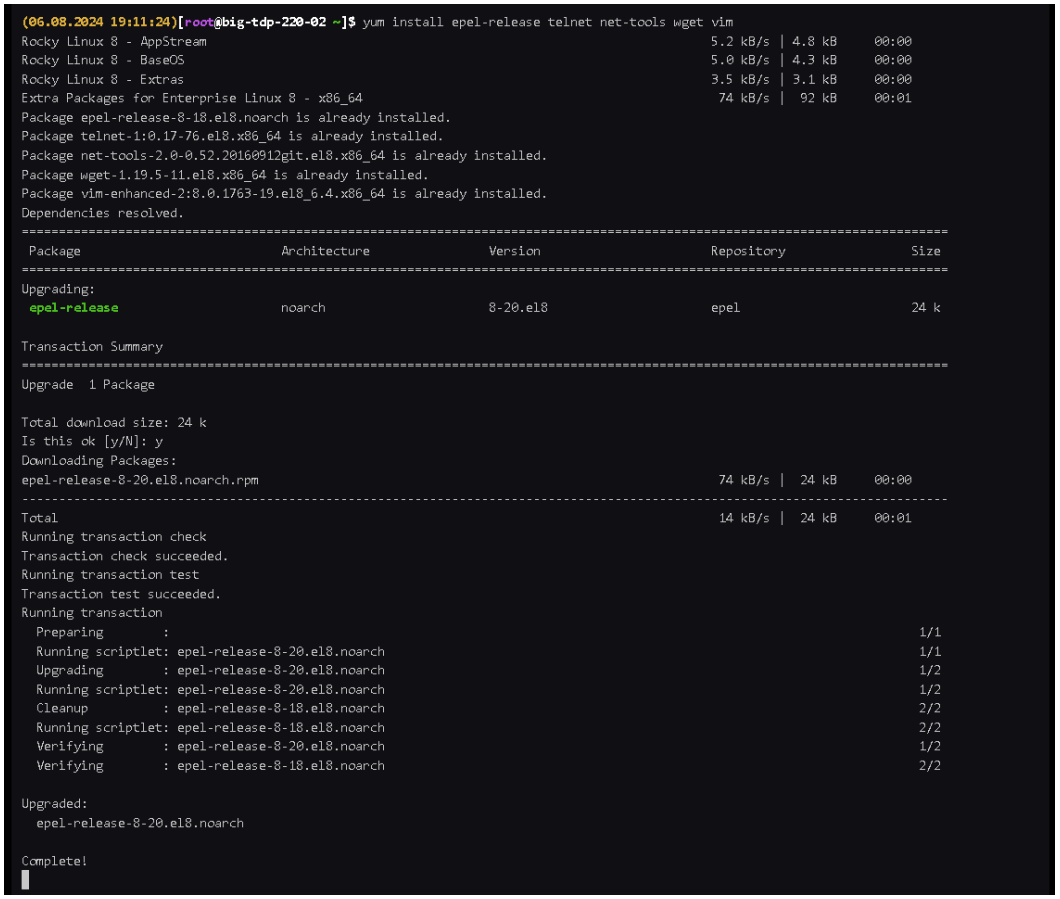


Figure 2 - Instalação das ferramentas de apoio

Environment Preparation

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;
```

> **AVISO**
>
> 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

```
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;
```

> **NOTA**
>
> O kernel do Linux disponibiliza uma configuração ajustável que controla a frequência de uso da área de swap em disco, 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.

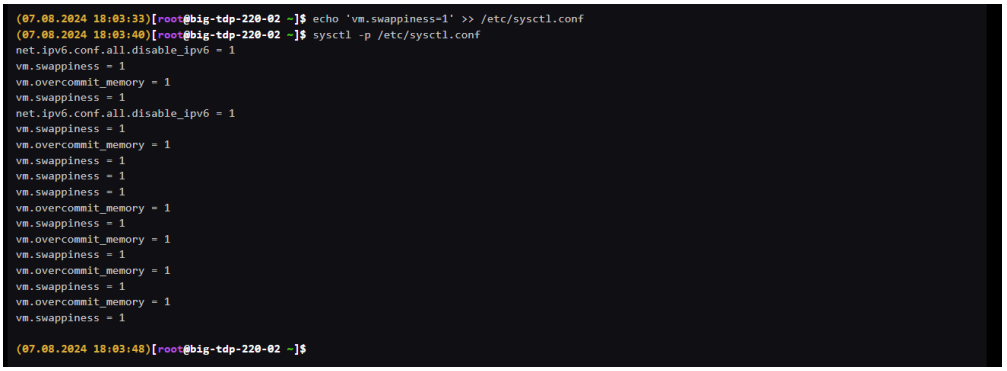


Figure 1 - Minimiza swap

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;
```

> **NOTA**
>
> 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):

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;
```

NOTA

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.

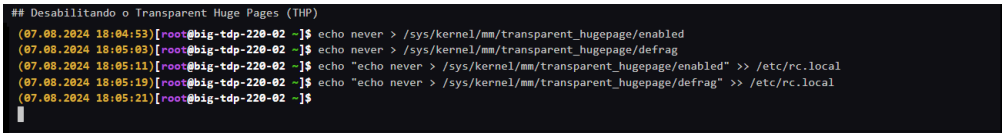


Figure 3 - Desabilita THP

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
```

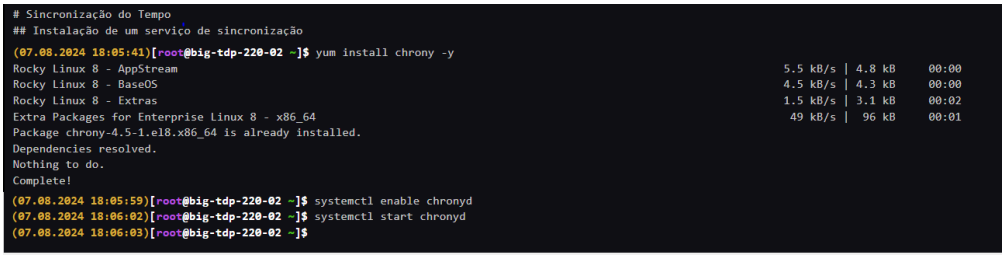


Figure 4 - Instala a sincronização do tempo

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
```

> **NOTA**
>
> 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
```

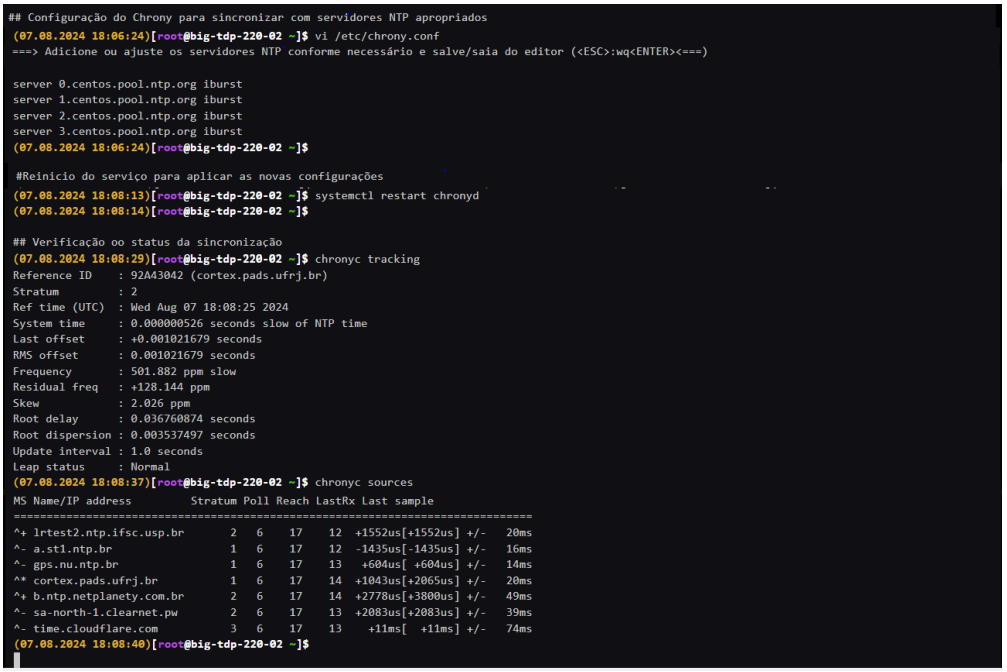


Figure 5 - Configura Chrony

Installation Packages

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.

> **NOTA**
>
> Em uma implantação com acesso limitado ou restrito à Internet, é possível realizar o download dos ficheiros 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 descarregar 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. Indique 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 pretendido para download dos ficheiros.

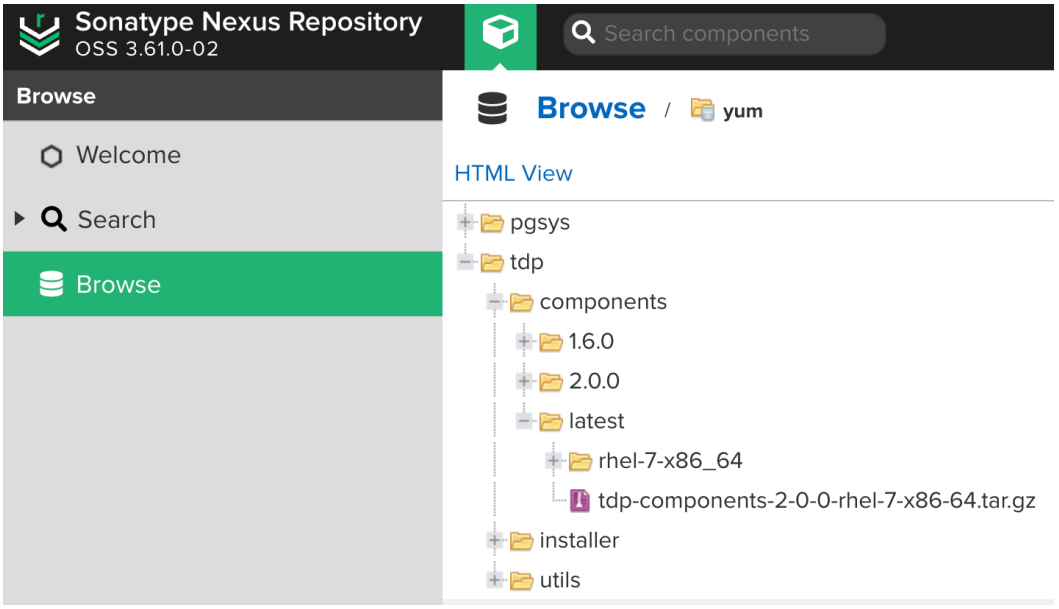


Figura 1 - Repositório Público de Pacotes da Tecnisys

Também é possível descarregar 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-2-2-el-9-x86-
64.tar.gz -o tdp-ambari-2-2-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-2-2-el-9-
x86-64.tar.gz -o tdp-components-2-2-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-2-2-el-9-x86-
64.tar.gz -o tdp-utils-2-2-el-9-x86-64.tar.gz
```

- Exemplo via wget

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/tdp-ambari-2-2-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-2-2-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-2-2-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 ficheiros compactados (tar.gz). Por exemplo:

Terminal input

```
/yum/tdp/ambari/3.0/tdp-ambari-2-2-el-9-x86-64.tar.gz
```

Terminal input

```
/yum/tdp/components/3.0/tdp-components-2-2-el-9-x86-64.tar.gz
```

Terminal input

```
/yum/tdp/utils/3.0/tdp-utils-2-2-el-9-x86-64.tar.gz
```

> **DICA**
>
> É possível verificar a integridade do ficheiro conferindo os atributos de Checksum (sha512, sha256, sha1 e md5), apresentados ao clicá-lo.

> **INFORMAÇÃO**
>
> Para uma instalação local, não se esqueça de descarregar 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 ficheiros compactados (tar.gz) dos pacotes de instalação de componentes e utilitários já contemplam a chave pública GPG.

Criação dum Repositório de Pacotes Local

Instruções

Numa 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 ficheiros 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
```

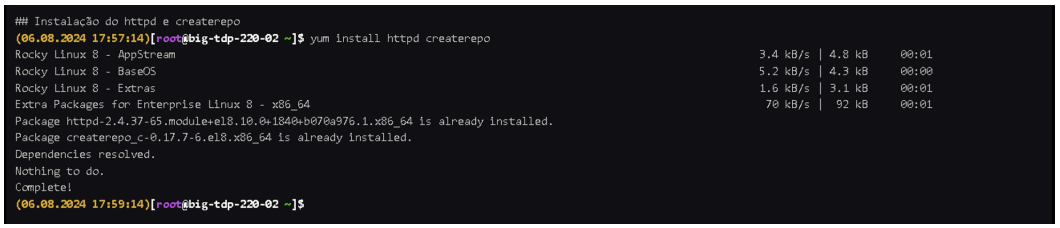


Figura 2 - Instalação createrepo e httpd

2. Crie o diretório do repositório de pacotes (ambari, components e utils). Por exemplo:

Terminal input

```
mv /tmp/tdp-ambari-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-ambari-2-2-el-9-x86-64.tar.gz -C /var/www/html
```

Terminal input

```
mv /tmp/tdp-components-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-components-2-2-el-9-x86-64.tar.gz -C /var/www/html
```

Terminal input

```
mv /tmp/tdp-utils-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-utils-2-2-el-9-x86-64.tar.gz -C /var/www/html
```

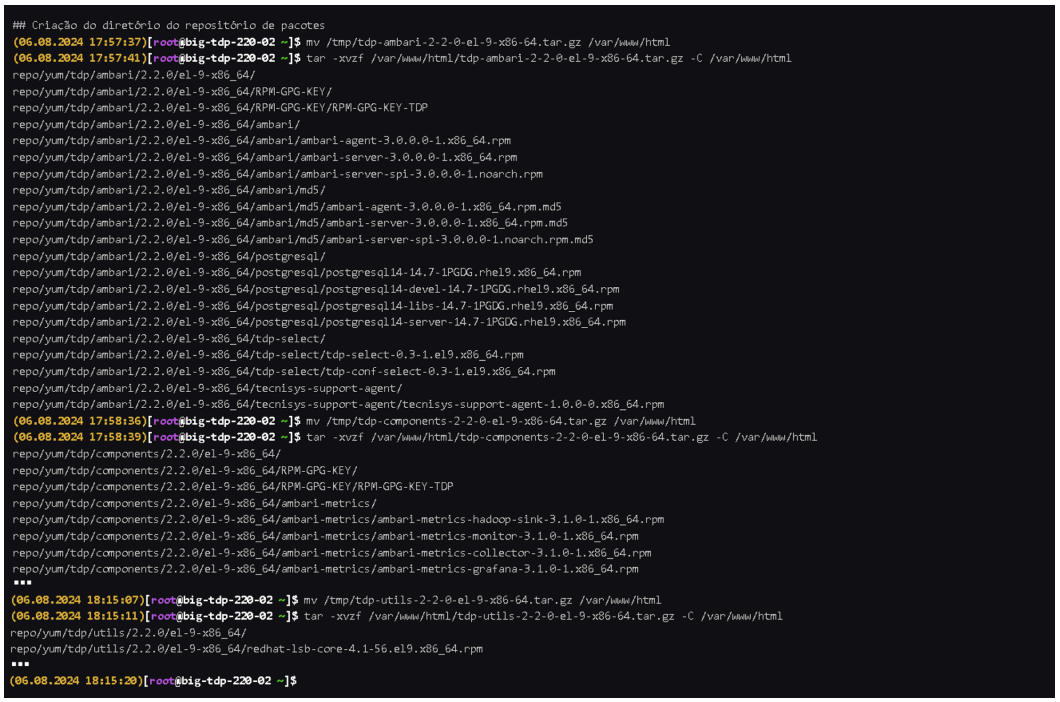


Figura 3 - Criação diretorio do repositorio

3. Crie o índice de pacotes do repositório:

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/ambari/3.0/el-9-x86_64/
```

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/
```

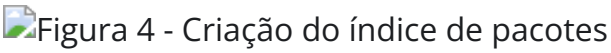


Figura 4 - Criação do índice de pacotes

4. Adicione o alias do repositório no ficheiro 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
```

Figura 5 - Adiciona alias do repositorio no ficheiro 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.

> **NOTA**
>
> Recomendamos que a instalação do Apache Ambari (Ambari Server e Ambari Web) seja realizada numa 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 registo das máquinas que irão compor o *Cluster de Big Data* venha a ser manual (sem a troca de chaves SSH), conforme descrito em Registo de Hosts, é recomendável que o script em questão seja baixado em todas as máquinas do *Cluster* para instalação do Ambari Agent.

É possível descarregar 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.

Apache Ambari Installation

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 numa 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.

> **NOTA**
>
> 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.

- **-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"
  ```

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.

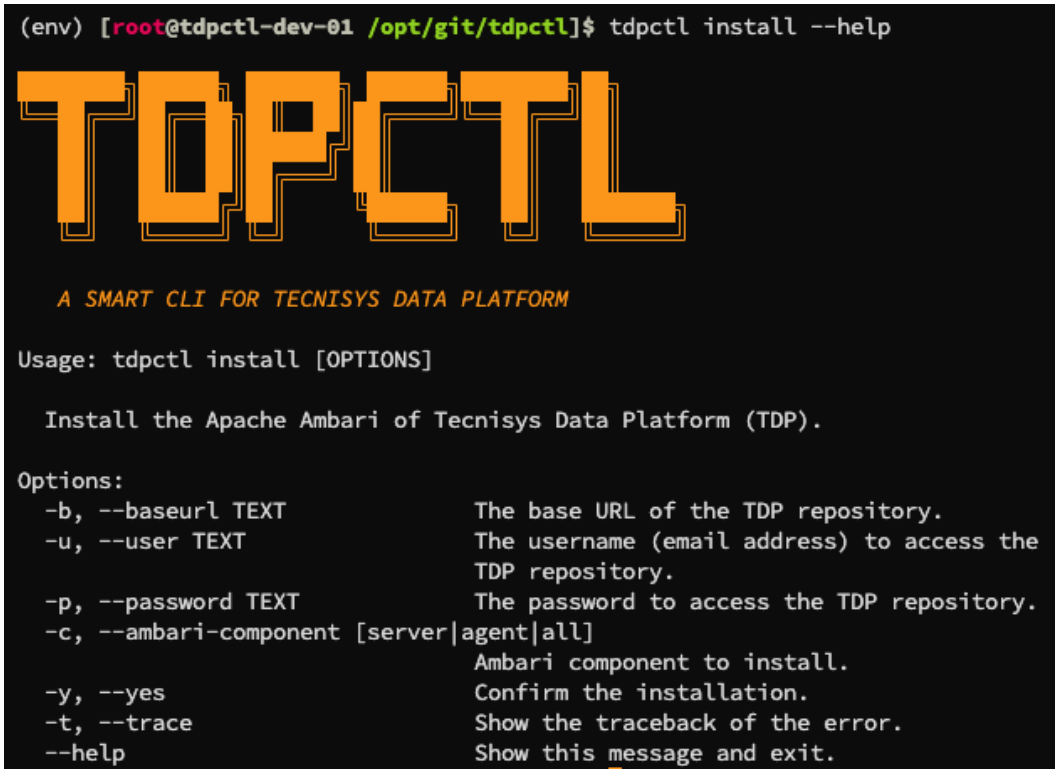


Figura 1 - Opções para o comando tdpctl install

Note que é possível informar o componente pretendido através da opção -c ou --component (agent | server | all).

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 na base de metadados:

Terminal input

```
ambari-server setup --jdbc-db=postgres --jdbc-driver=postgresql-42.2.16.jar
```

> **DICA**
>
> O driver JDBC do PostgreSQL também está disponível no diretório public/ambari/3.0.0.0 do Repositório de Pacotes Tecnisys

Figura 3 - Configuração do JDBC Server

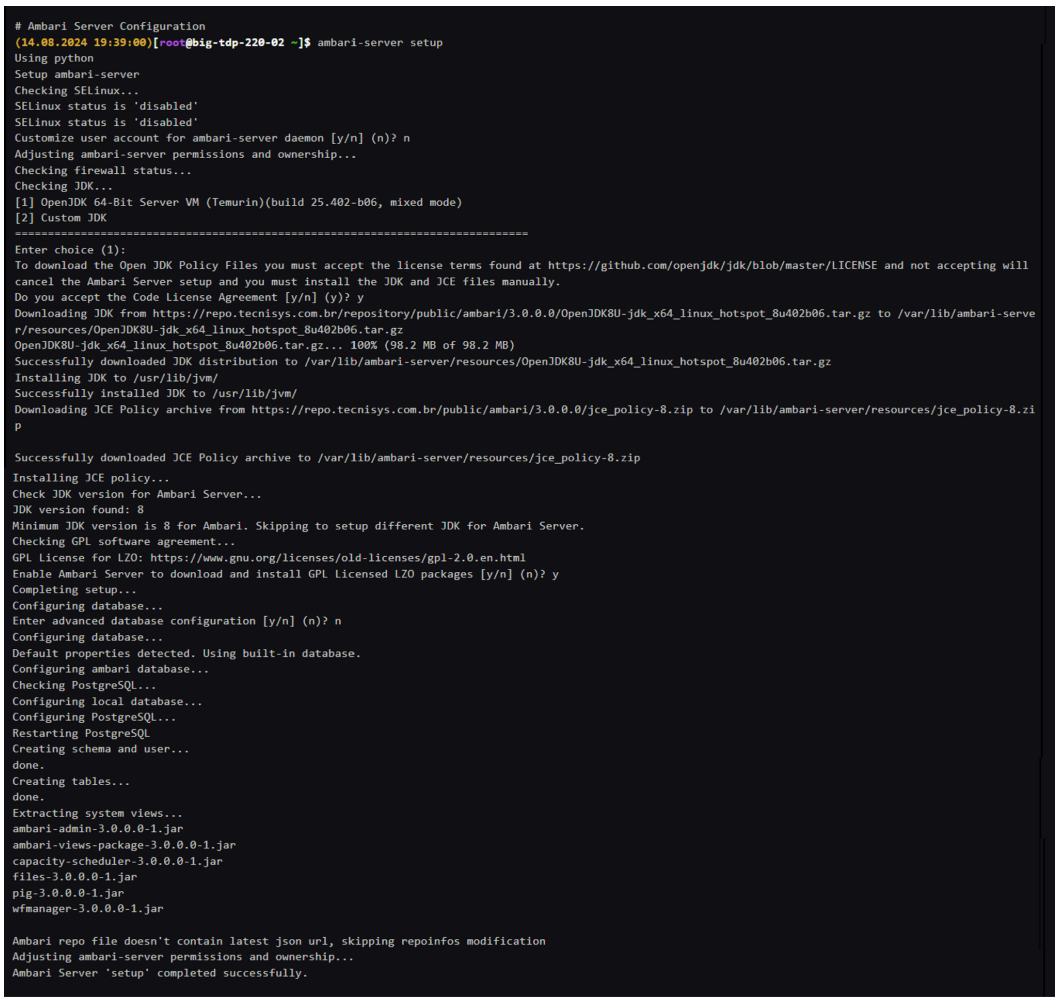


Figura 4 - Configuração do Ambari Server

Configuração do Ambari Server

Configure o Ambari Server:

Terminal input

```
ambari-server setup
```

De seguida, 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), indique 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 Base 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 própria base de metadados, esquema, usuário, tabelas de metadados, etc.

> (b) - Se sim (y), serão solicitadas informações de conexão aa base de dados, tais como, o Sistema Gerenciador de Base de Dados (SGBD), hostname, porta e nome da base de dados do Ambari, para que o Ambari Server se conecte e crie automaticamente as suas tabelas de metadados entre outras estruturas.

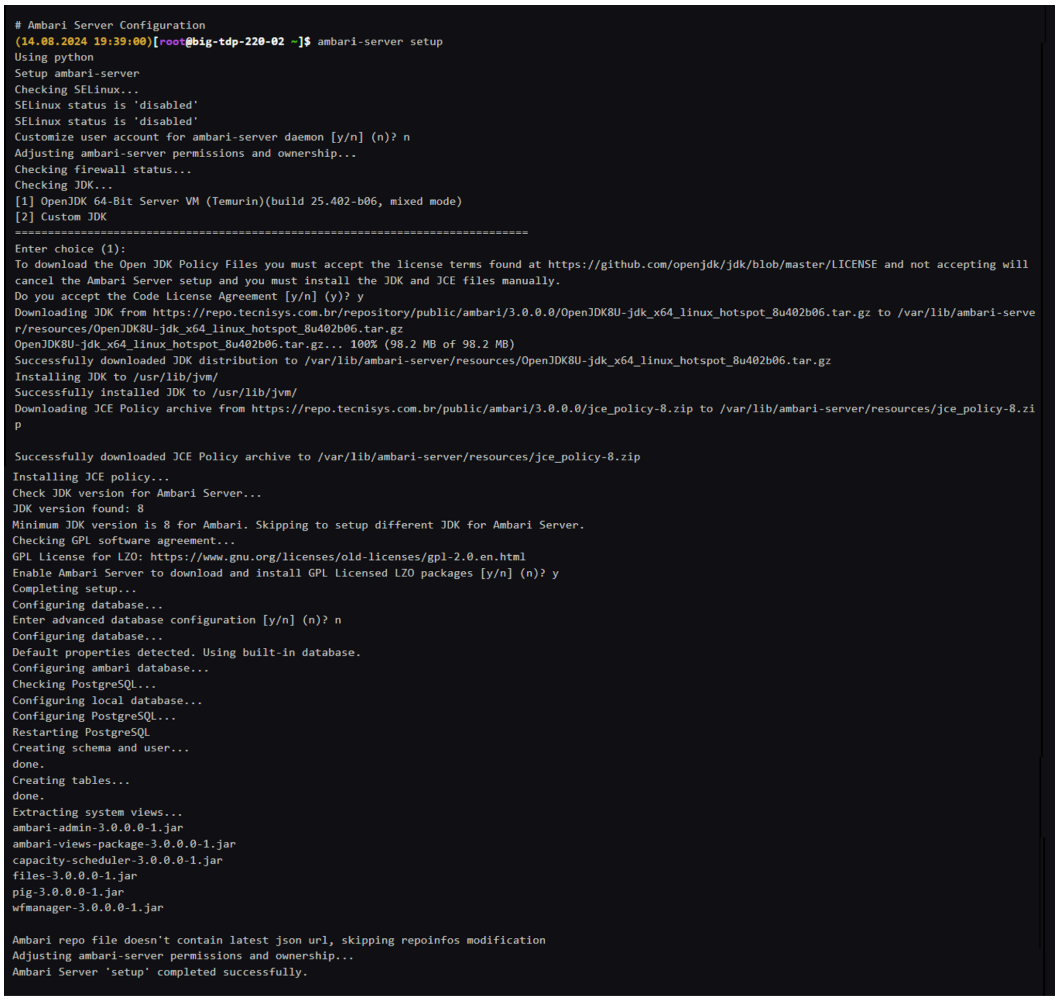


Figure 4 - Configuração do Ambari Server

Inicialização do Ambari Server

Inicie o daemon do Ambari Server:

Terminal input

```
ambari-server start
```

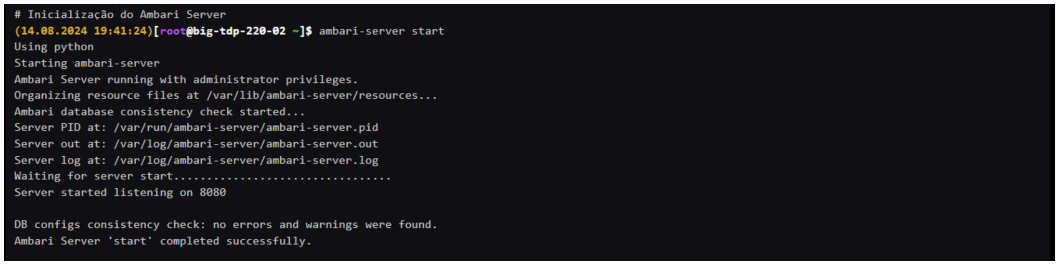


Figura 5 - Inicialização do Ambari Server

Após a inicialização do Ambari Server, a página web do Ambari estará, por padrão, acessível na porta 8080.

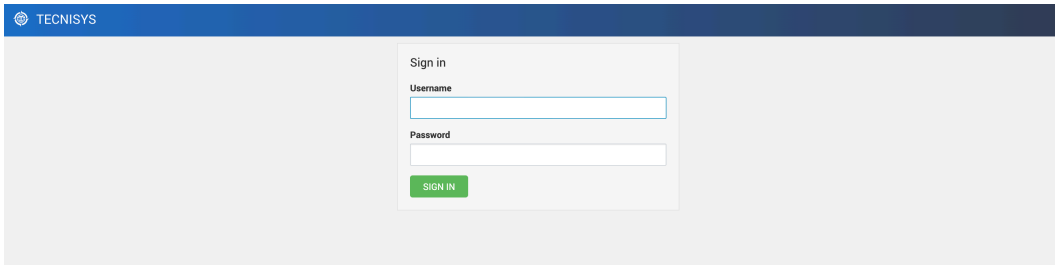


Figura 6 - Página de login do Ambari

> **NOTA**
>
> Se necessário, a porta da interface web do Ambari pode ser alterada no ficheiro `/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, disponibilizamos 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.

> **AVISO**
>
> As senhas utilizadas nos comandos a seguir visam apenas exemplificar o procedimento apresentado. Por gentileza, utilize senhas fortes em seu

ambiente. Além disso, os nomes dos usuários e bancos de dados são apenas sugestões, fique à vontade para alterá-los.

ATENÇÃO

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 ordenação 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 a base 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:

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 a base 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 a base de dados hive:

Terminal input

```
su -c 'createdb -O hive hive' - 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 a base de dados ranger:

Terminal input

```
su -c 'createdb -O ranger ranger' - postgres;
```

Criação do Banco de Metadados do Apache RangerKMS

1. Crie o usuário ranger:

Terminal input

```
su -c 'createuser -s -d -l rangerkms' - postgres;
```

2. Defina a senha do usuário rangerkms:

Terminal input

```
su -c "psql -U postgres -c \"alter user rangerkms with password
'rangerkms'\"" - postgres;
```

3. Crie a base de dados rangerkms:

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 a base de dados superset:

Terminal input

```
su -c 'createdb -O superset superset' - postgres;
```

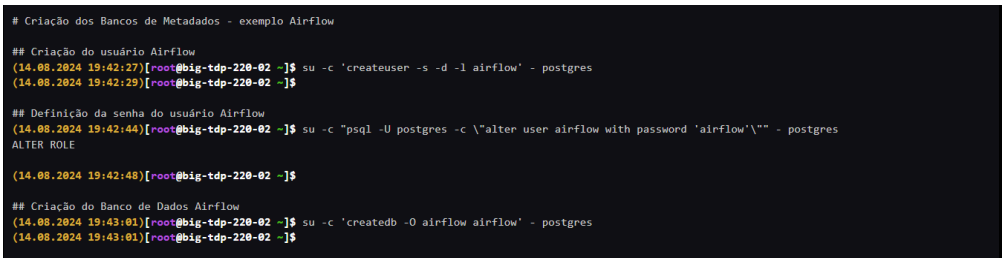


Figura 7 - Criação do metadados

Configuração da Instância dos Bancos de Metadados

A seguir, disponibilizamos 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.

> **NOTA**
>
> Verifique, e se necessário altere, a versão majoritária do PostgreSQL presente nos caminhos dos ficheiros de configuração.

1. Adicione os demais bancos de dados à regra de acesso do Ambari :

Terminal input

```
sed -i
's/ambari,mapred/airflow,ambari,druid,hive,mapred,oozie,ranger,superset/g'
/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:

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
```

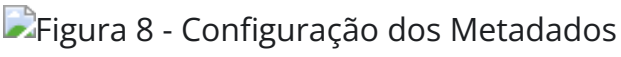


Figura 8 - Configuração dos Metadados

> **AVISO**
>
> Avalie os impactos de tais configurações na segurança e desempenho do seu ambiente. Caso necessite de apoio, Entre em contacto, 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. ↩

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 navegador para aceder a interface web do Ambari disponível no IP/hostname da máquina do Ambari Server, porta 8080. Por exemplo:

```
http://192.168.56.100:8080
```

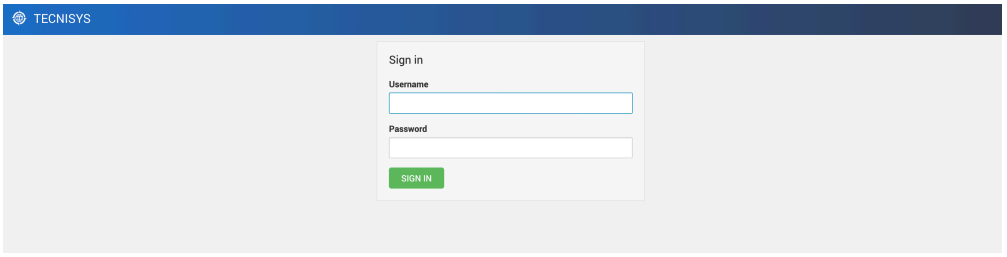


Figura 1 - Página de login do Ambari

> **NOTA**
>
> 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:

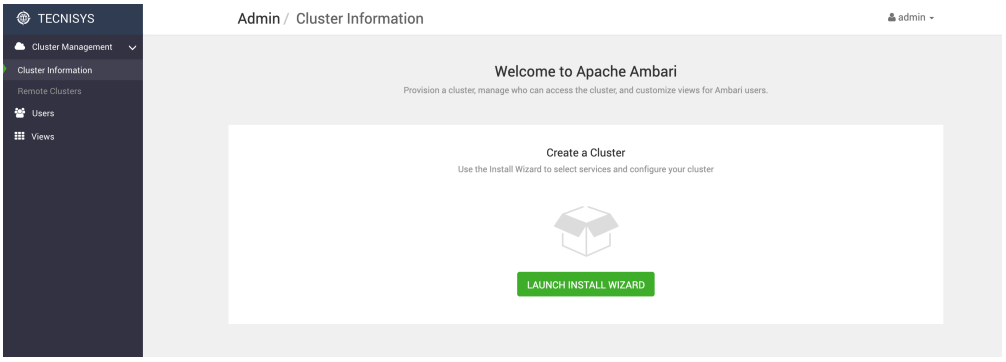


Figura 2 - Página de Boas Vindas do Ambari

2. Indique um nome para o Cluster e clique em NEXT:

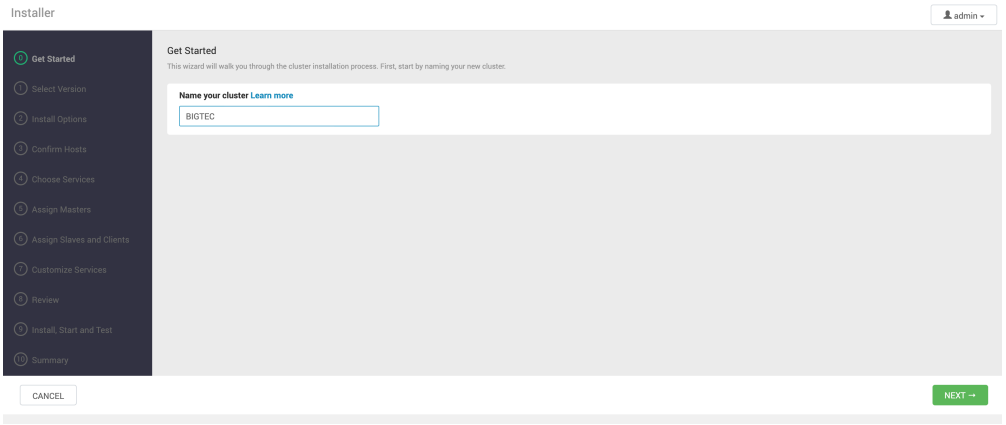


Figura 3 - Nome do Cluster

Seleção da Versão

1. Selecione a versão do TDP desejada:

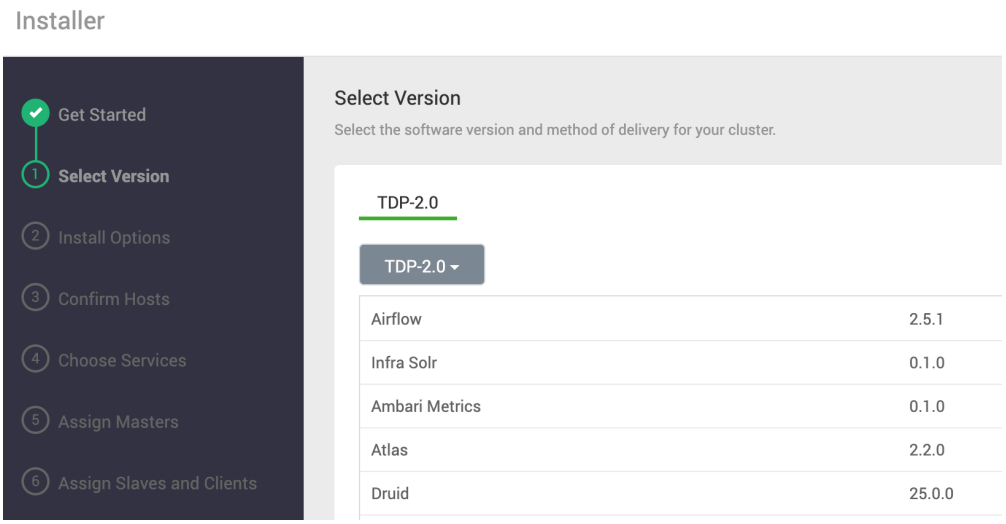


Figura 4 - Versão do TDP

2. Selecione o tipo do repositório de pacotes (Public ou Local) e indique a URL para *Components* (TDP-3.0) e *Utils* (TDP-UTILS-3.0):

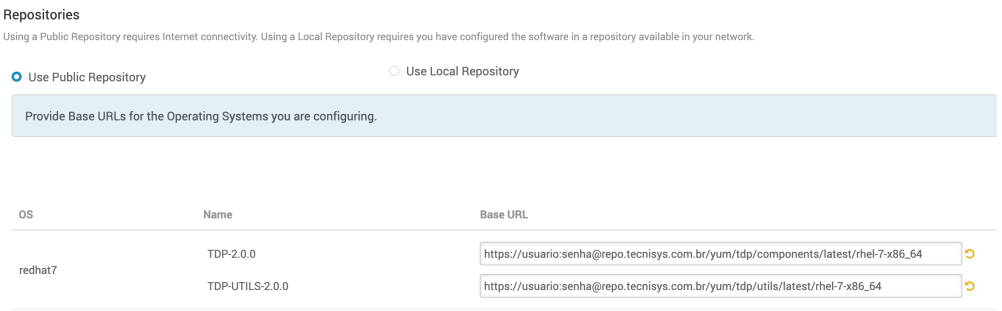


Figura 5 - Repositórios de pacotes

2.1. De seguida, 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, indique 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 dum Servidor de DNS (recomendado) ou localmente (ficheiro `/etc/hosts`).

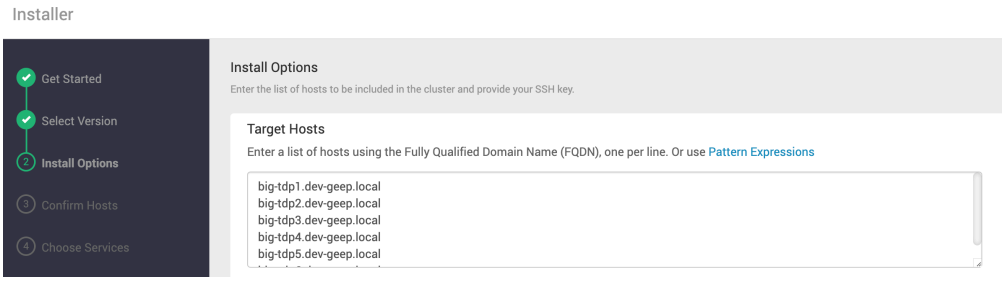


Figura 6 - Informação dos hosts do Cluster

> **DICA**

> 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 registadas.

    - Caso opte por disponibilizar a chave privada SSH da máquina do Ambari Server para o registo automático das máquinas do *Cluster*, cole o seu conteúdo no campo de texto abaixo ou faça o *upload* de seu ficheiro. De seguida, 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, aceder todas as máquinas via SSH sem a informação da senha do usuário do *daemon* do Ambari Server (por padrão, root).

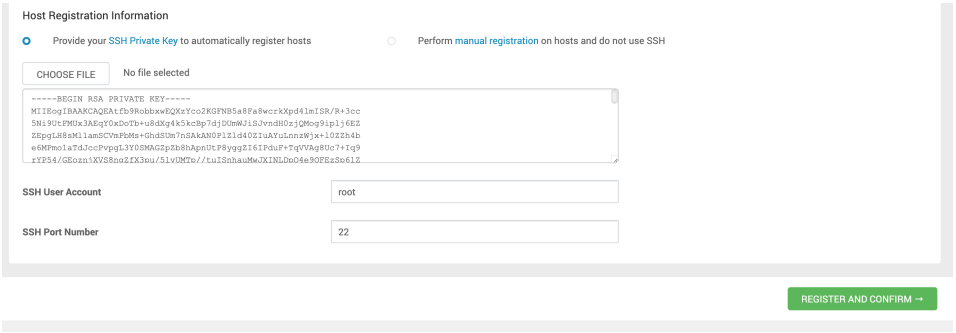


Figura 7 - Registo dos hosts

    - Caso opte por realizar o registo manual das máquinas, faça você mesmo a instalação do Ambari Agent em todas as máquinas antes de prosseguir.

> **DICA**
>
> A chave privada SSH do tipo RSA pode ser obtida executando o seguinte comando:
>
> **Terminal input**
>
> ```
> cat ~/.ssh/id_rsa
> ```

> **NOTA**
>
> Para instalar manualmente o Ambari Agent:
>
> Terminal input
>
> ```
> yum install ambari-agent
> ```

3. De seguida, clique em REGISTER AND CONFIRM.

Configuração da Relação de Confiança

1. Na máquina do Ambari Server, gere uma chave privada SSH:

   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).

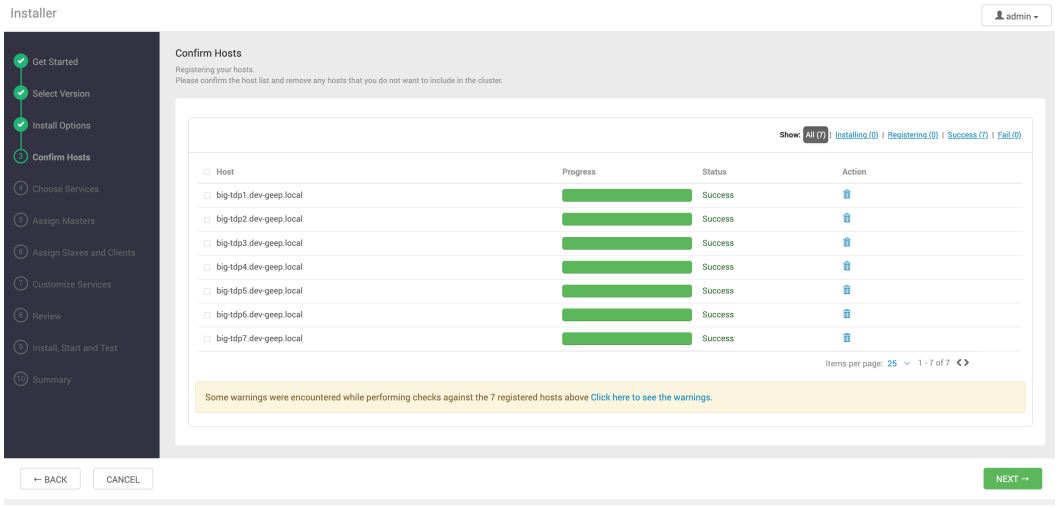


Figura 8 - Confirmação dos hosts

Eventuais erros precisam ser corrigidos e a verificação reexecutada para prosseguir.

Clique em NEXT para avançar.

> **NOTA**
>
> 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.

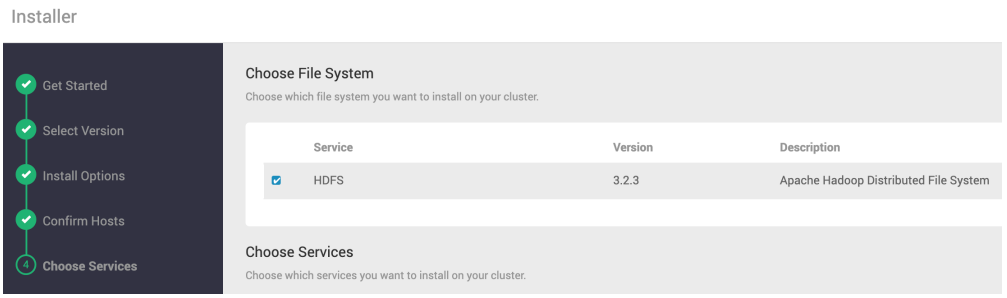


Figura 9 - Seleção do serviço da camada de armazenamento

2. Selecione os demais serviços do Cluster.

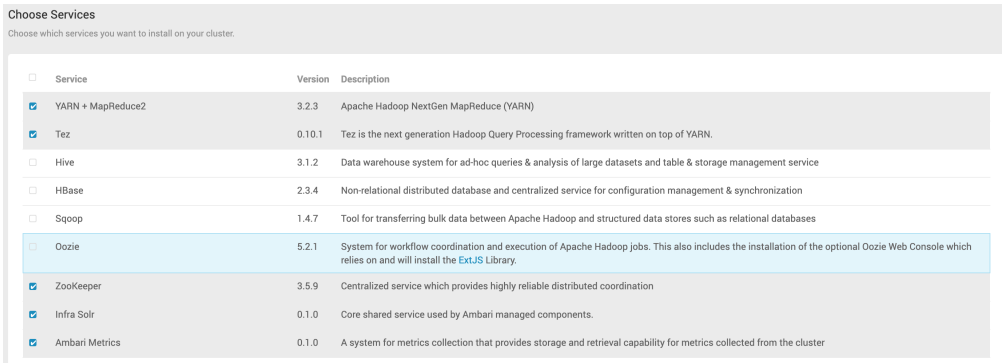


Figura 10 - Seleção dos demais serviços do Cluster

> **DICA**
>
> 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.

> **NOTA**
>
> 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 dum 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. De seguida, 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.

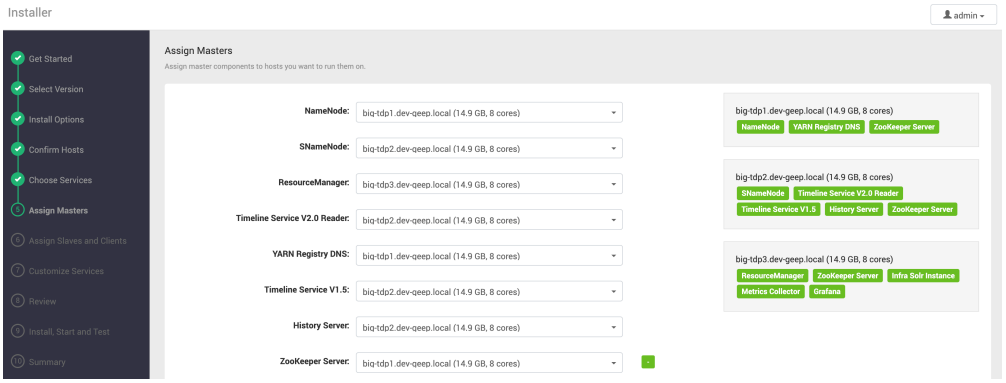


Figura 11 - Atribuição dos componentes Masters

> **NOTA**
>
> 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. De seguida, 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.

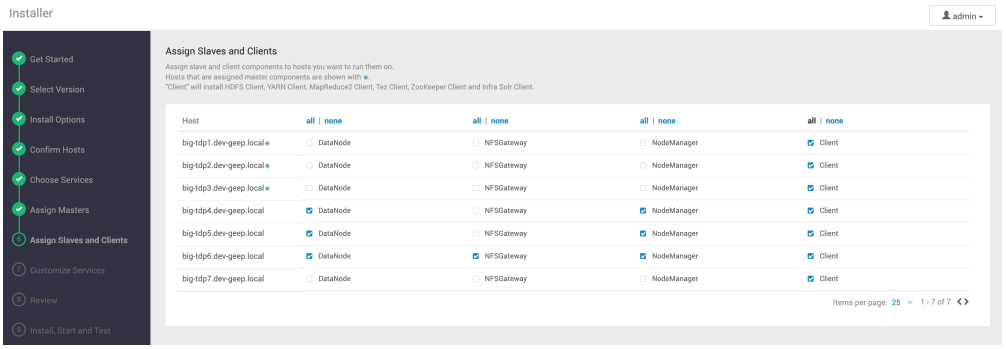


Figura 12 - Atribuição dos componentes Slaves e Clients

> **NOTA**
>
> Sempre que possível, evite instalar componentes Slaves em máquinas de componentes Masters.

2. De seguida, 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:

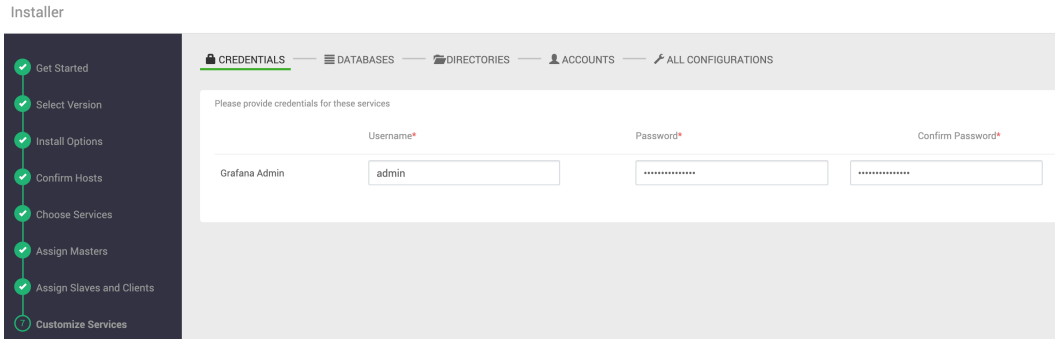


Figura 13 - Definição das credenciais de administração do Grafana

Bancos de Dados

Ilustrando essa seção, temos o Hive, o qual requer uma base de dados para persistência de metadados. Nesse exemplo, informamos os dados de conexão para uma instância PostgreSQL existente:

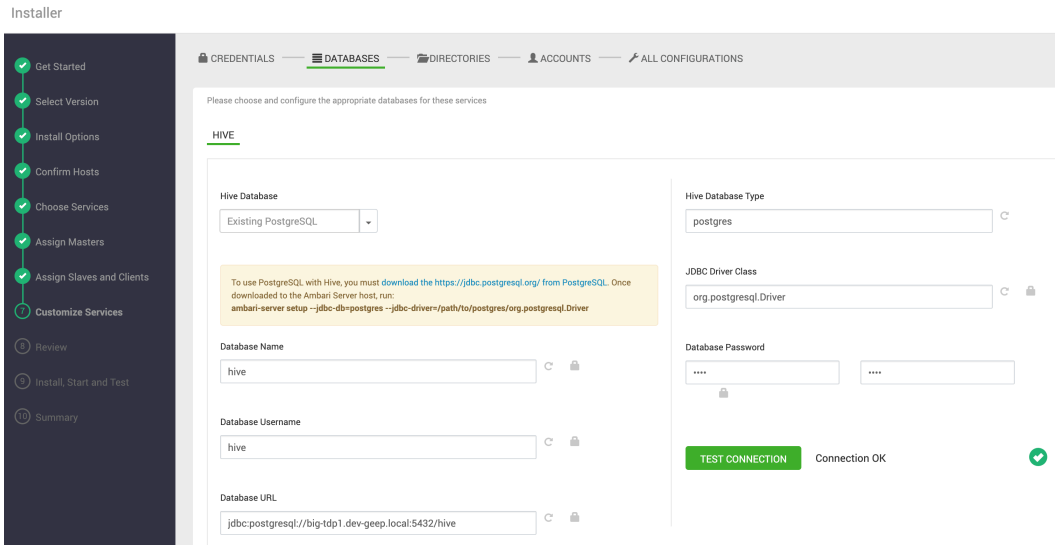


Figura 14 - Definição dos dados de conexão da Base de Dados do Hive

> **NOTA**
>
> Clique no botão TEST CONNECTION para testar a conexão com a Base 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.

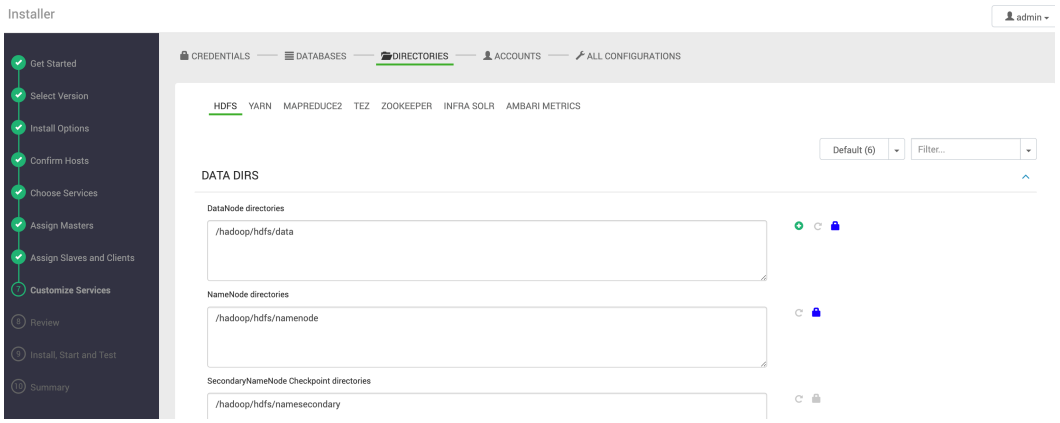


Figura 15 - Definição dos diretórios dos serviços

> **AVISO**
>
> Se possível, use dispositivos de armazenamento (discos, SSD, entre outros), volumes e diretórios exclusivos para os ficheiros 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.

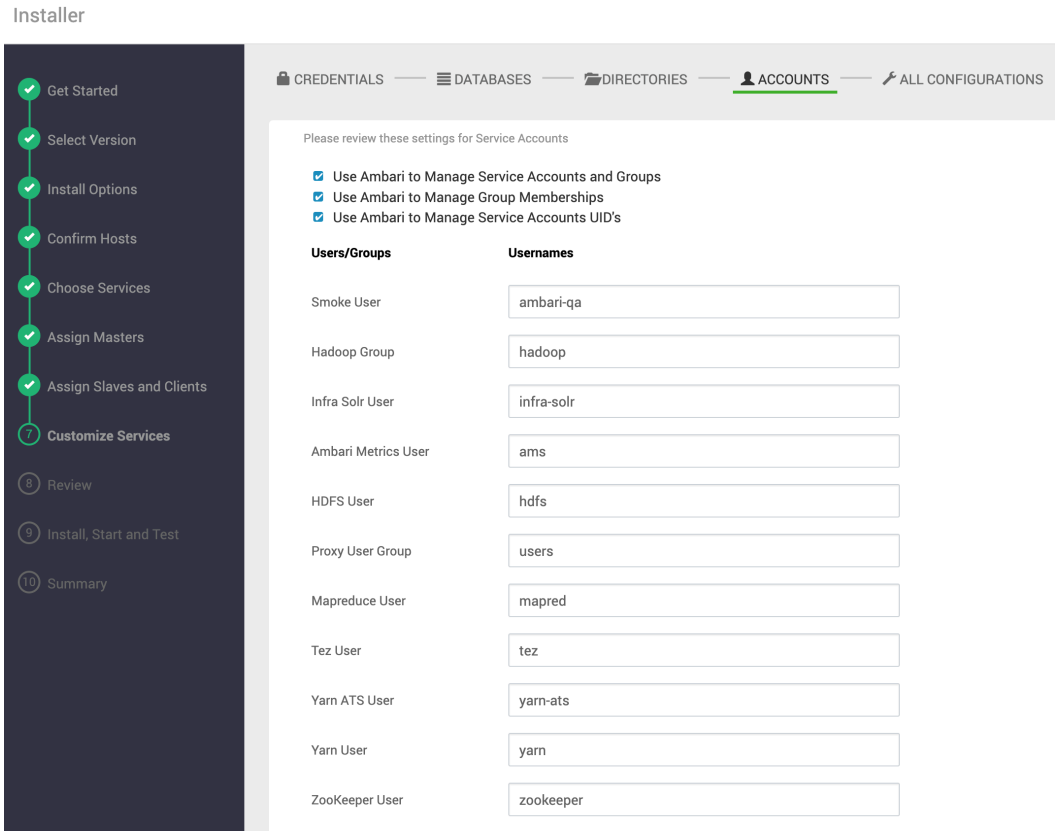


Figura 16 - Definição dos usuários dos serviços

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.

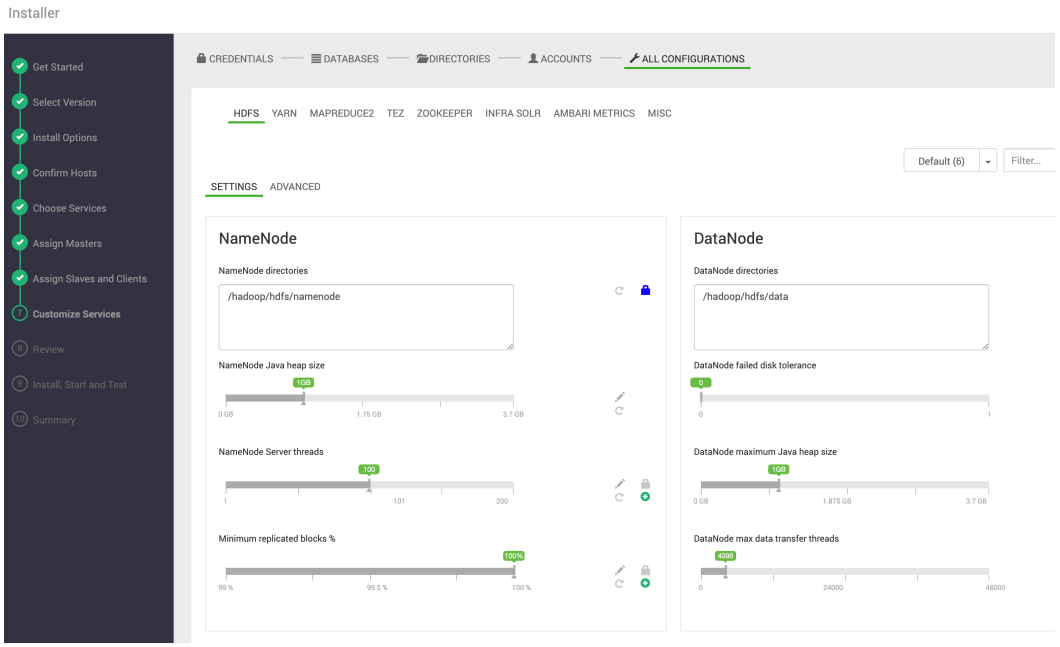


Figura 17 - Todas as configurações dos serviços

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.

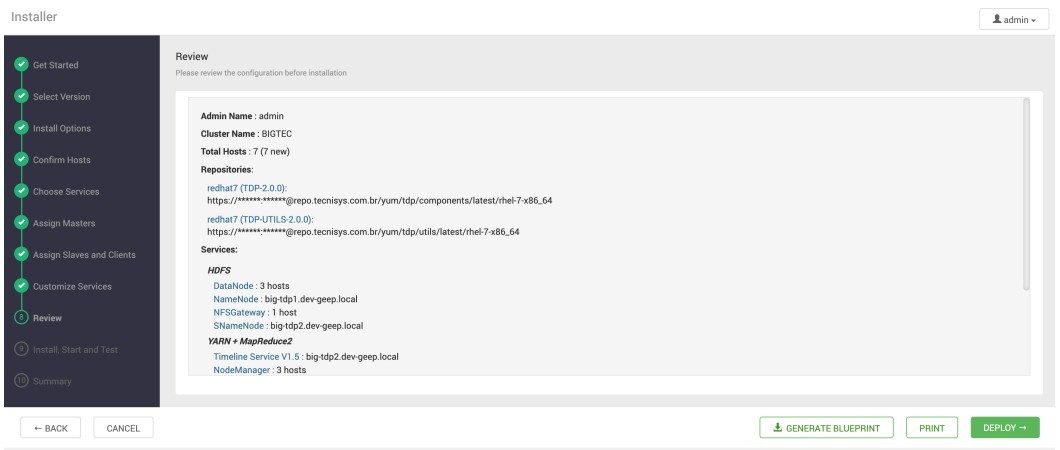


Figura 18 - Revisão da configuração antes da instalação

> **DICA**

Utilize o botão PRINT para gerar um relatório da instalação e o botão GENERATE BLUEPRINT para gerar um ficheiro 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.

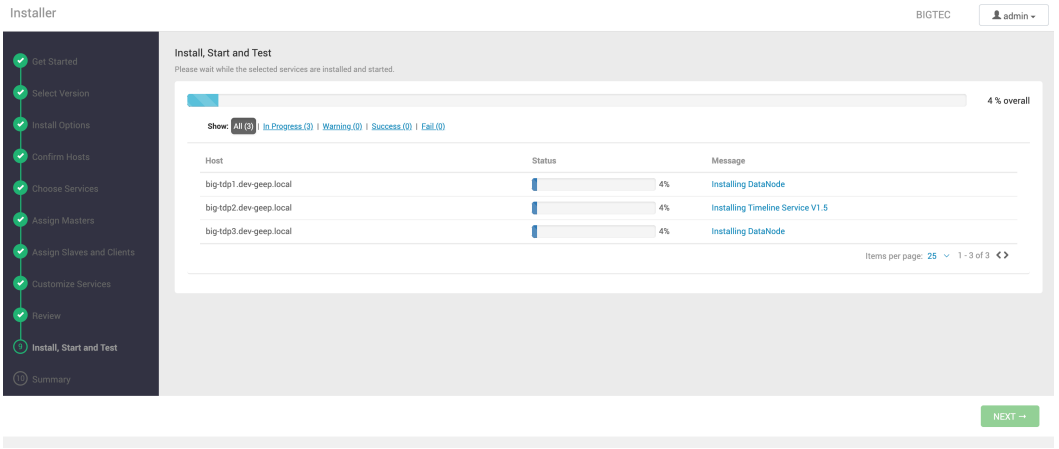


Figura 19 - Instalação dos serviços

NOTA

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 dum 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.

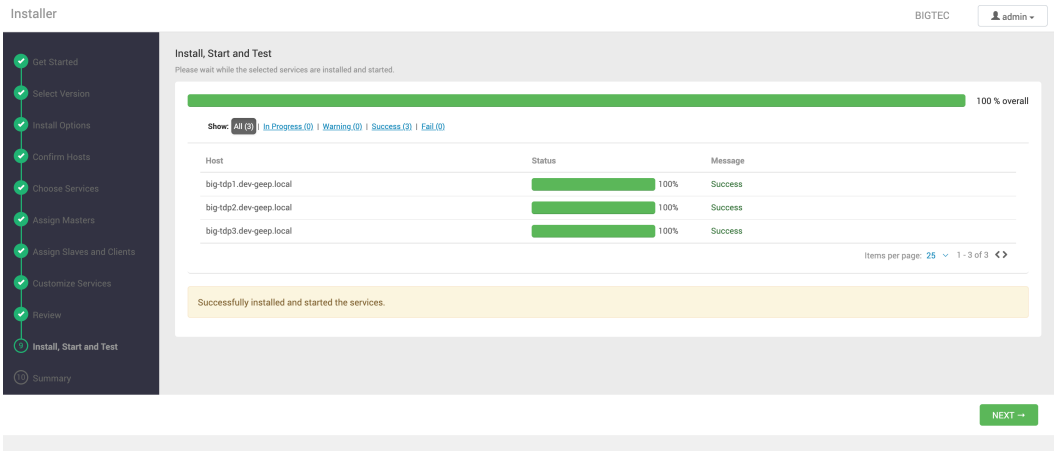


Figura 20 - Instalação, Iniciação e Testes finalizados

Sumário

Na última etapa do processo é apresentado um resumo da implementação.

Clique no botão COMPLETE para finalizar a operação e aceder a área de administração do *Cluster* criado.

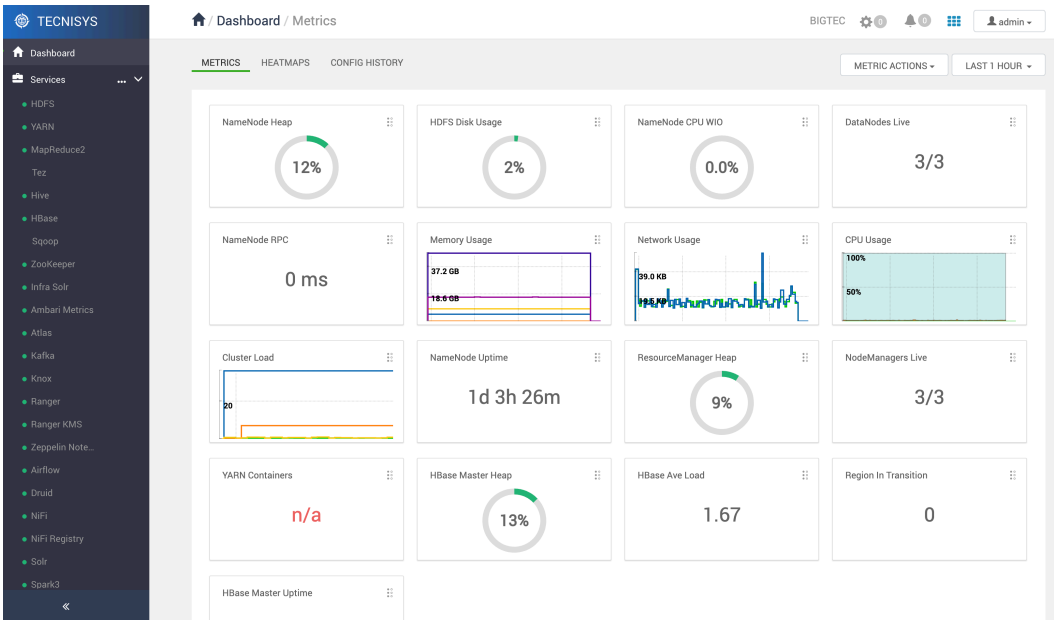


Figura 21 - Área de administração do Cluster

PARTE IV - ATUALIZAÇÃO

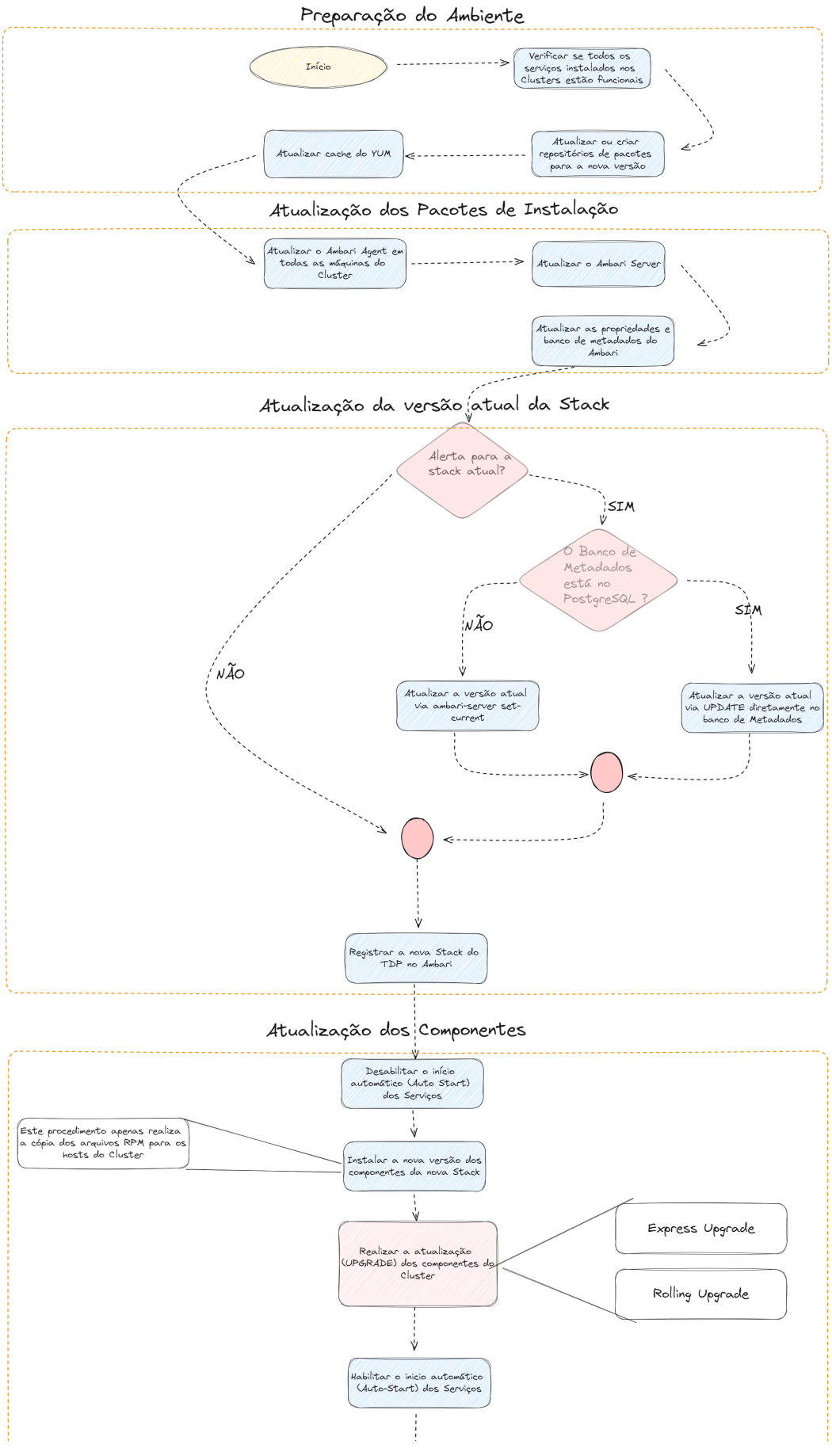
Fluxograma de Atualização

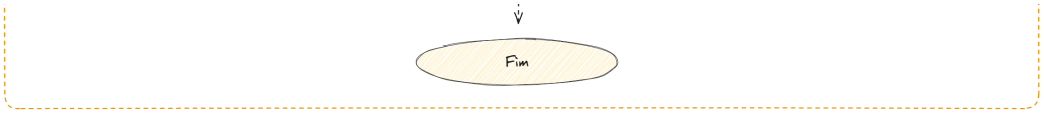
O fluxograma a seguir representa graficamente o processo de atualização dum Cluster TDP.

Resumidamente, o processo de atualização do TDP se divide em três etapas principais:

- Atualização do Apache Ambari;
- Registo da nova versão da *stack* do TDP; e
- Atualização dos componentes instalados no *Cluster*.

Nas próximas seções, detalharemos cada uma dessas etapas.





Pré-Requisitos para Atualização

A seguir apresentamos os pré-requisitos mínimos para a atualização dum *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, ficheiros de configuração e bibliotecas. Para uma atualização segura, sem intercorrências, recomendamos um mínimo de 100 GB de espaço livre em disco.

> **AVISO**
>
> Antes da instalação dos pacotes da nova versão, verifique o espaço em disco disponível, evitando assim a corrupção de ficheiros, 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`.

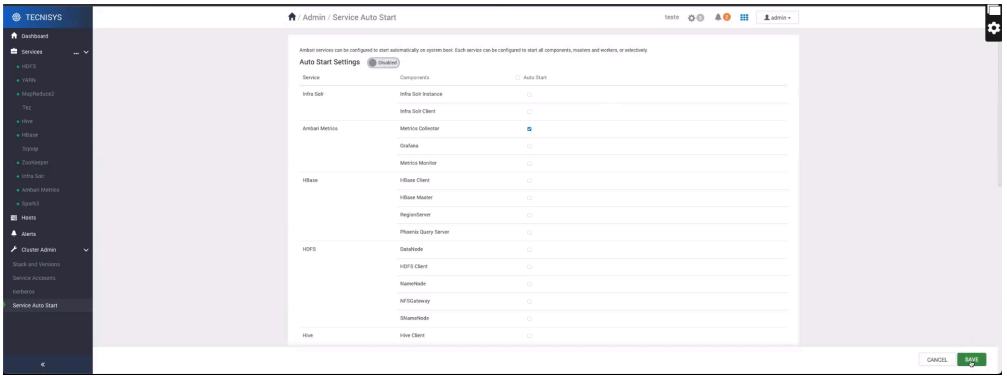


*Figura 5 - Desativando o _Auto Start_*

- Clique em `Save` para salvar a alteração e confirme a operação.

> **NOTA**
>
> 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 dum 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

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 ficheiro de configuração `/etc/ambari-server/conf/ambari.properties` da máquina do Ambari Server. Para isso, siga os passos a seguir:

Instruções

1. Abra o ficheiro 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 ficheiro de configuração do Ambari Server.

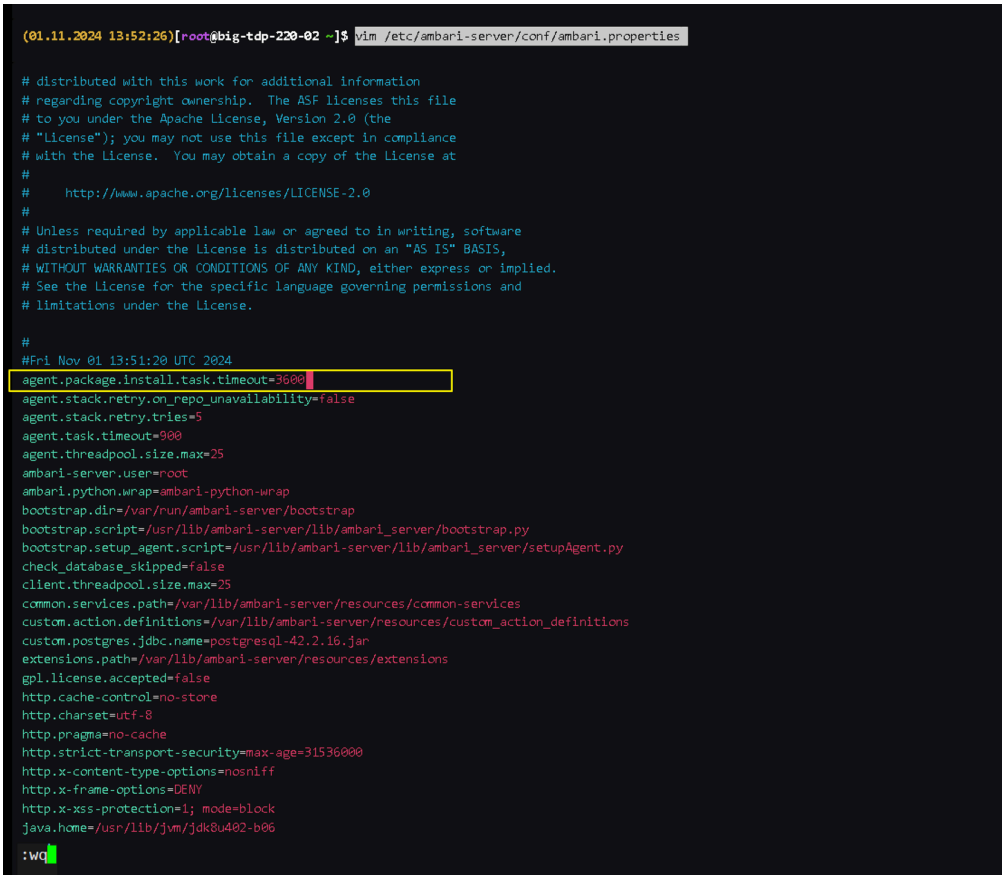


Figura 6 - Ajuste timeout

2. Reinicie o serviço do Ambari Server:

Terminal input

```
ambari-server restart
```

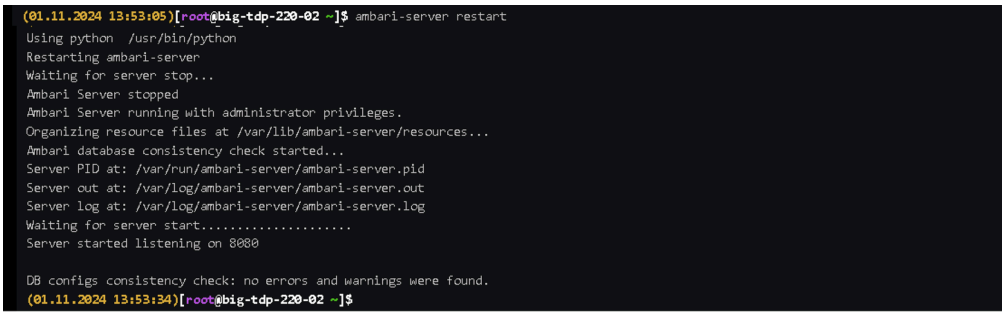


Figura 6 - Ajuste timeout

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

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 ficheiro `/etc/ambari-server/conf/ambari.properties` da máquina do Ambari Server.

> **NOTA**
>
> Caso o parâmetro `upgrade.parameter.nn-restart.timeout` não exista no ficheiro 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 ficheiro 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 ficheiro de configuração do Ambari Server.

2. Reinicie o serviço do Ambari Server:

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

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 dum novo repositório de pacotes para o Apache Ambari:

- Acesse o diretório em que se encontra o ficheiro do repositório de pacotes (*repo file*) do Ambari:

  Terminal input

  ```
  cd /etc/yum.repos.d/
  ```

- Execute o comando abaixo para descarregar o ficheiro `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-
  ```

```
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
```

> **AVISO**
>
> Os comandos sugeridos abaixo sobrescrevem o ficheiro `/etc/yum.repos.d/ambari.repo`, caso ele já exista no diretório. Se necessário, faça antes um *backup* do ficheiro com as definições atuais.

- No ficheiro 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 fast;
```

Atualização do Ambari Agent

Instruções

Com o novo repositório de pacotes definido, inicie a atualização do Ambari atualizando o componente Ambari Agent.

> **AVISO**
>
> A comunicação entre o Ambari Agent e o Ambari Server é suspensa durante a atualização. Entretanto, os serviços dos demais componentes do *Cluster*

continuam em execução.

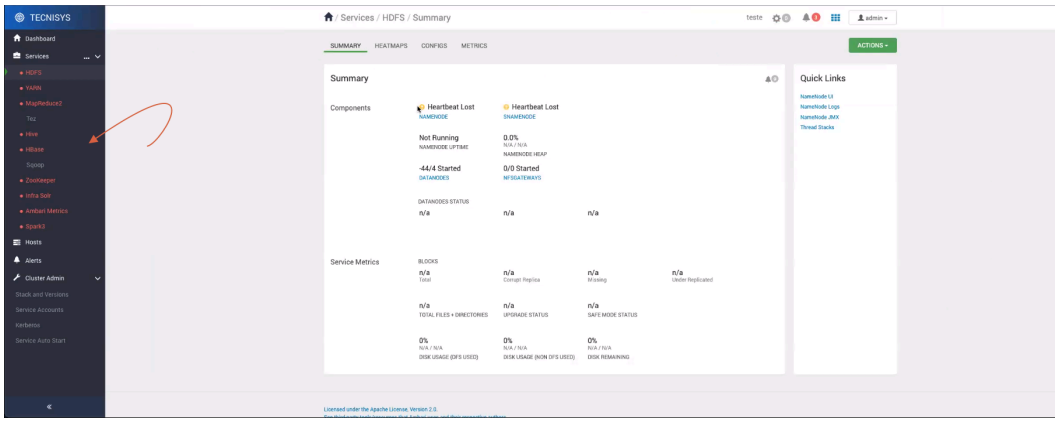


*Figura 1 * Situação dos serviços durante a atualização do Ambari Agent*

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
```

> **NOTA**
>
> Eventuais customizações do ficheiro `ambari-agent.ini` podem ser recuperadas no ficheiro de backup gerado: `ambari-agent.ini.old`

- Inicie o serviço do Ambari Agent:

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.

> **AVISO**
>
> Essa operação requer a parada do serviço do Ambari Server.

> **AVISO**
>
> É recomendada a realização dum *backup* da base 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
```

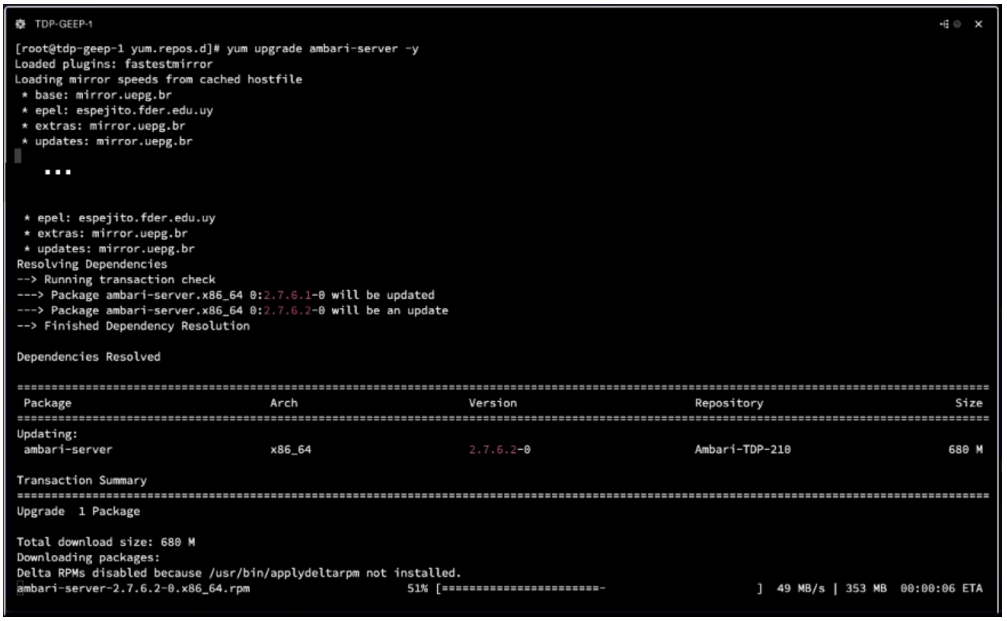


Figura 2 - Atualização dos binários do Ambari Server

- Atualize os metadados do Ambari Server:

Terminal input

```
ambari-server upgrade
```

> **NOTA**
>
> Basicamente, este comando realiza as seguintes operações:
>
> - Ajustes de propriedades do Ambari.
> - Atualização do *schema* da base de metadados.
> - Verificação de novas versões da *stack* TDP.

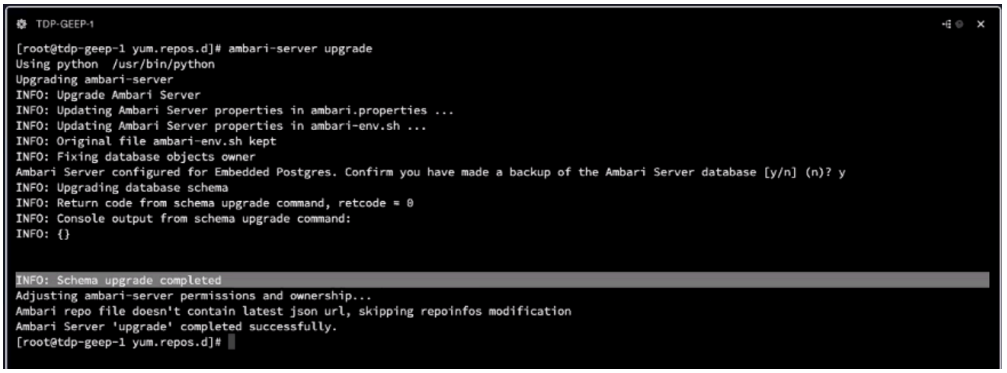


Figura 3 - Ambari Server Upgrade

- 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*.

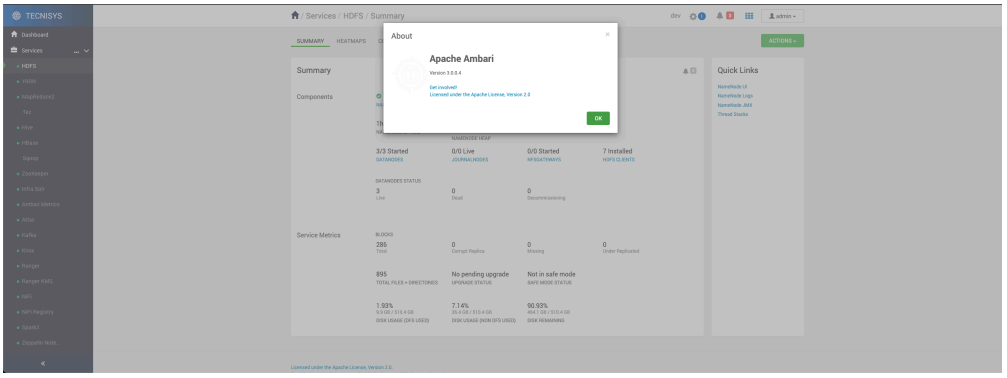


Figura 4 - Versão do Ambari

- 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.

Atualização da Versão Atual da *Stack*

Instruções

Caso seja exibido o alerta ´Host component out of sync´ ao verificar as versões da *stack* do TDP na interface web do Ambari, realize o procedimento detalhado nesta seção para corrigir essa inconsistência eventual entre a versão atual da *stack* e a informação definida na base de metadados do Ambari.

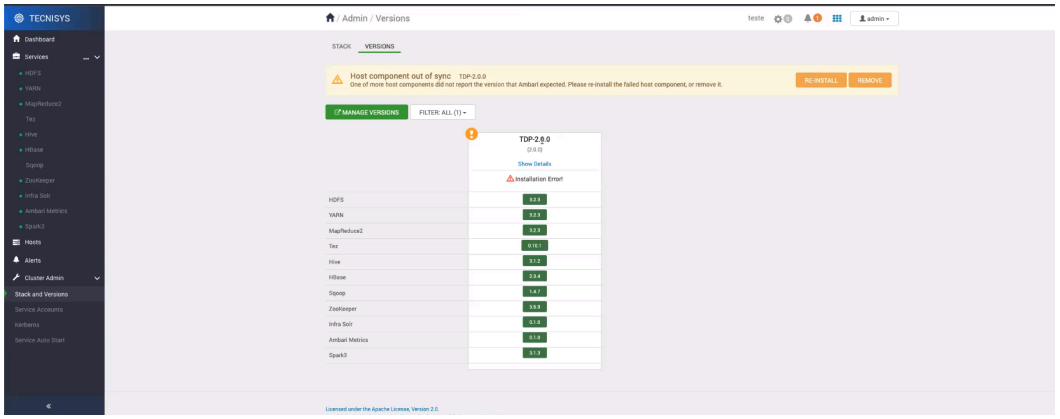


*Figura 5 - Alerta na guia _VERSIONS_*

> **AVISO**
>
> Essa operação requer a parada do serviço do Ambari Server.

> **AVISO**
>
> É recomendada a realização dum *backup* da base de metadados do Ambari antes do início desta operação.

Execute o procedimento a seguir na máquina em que está instalado o Ambari Server:

- Pare o Ambari Server:

Terminal input

```
ambari-server stop
```

- Atualize a base de metadados do Ambari.

Se a base de metadados NÃO está no PostgreSQL, acesse a Base de Dados em questão e execute o seguinte comando:

Terminal input

```
UPDATE ambari.host_version SET state = ´CURRENT´ WHERE state!= ´CURRENT´;
```

Se a base de metadados está no PostgreSQL:

Terminal input

```
    ambari server set current --cluster name <NOME-DO-CLUSTER> --version display
name TDP<VERSÃO>
```

Figura 6 - Definição da versão corrente da stack

> **NOTA**
>
> Certifique-se que o NOME-DO-CLUSTER e a VERSÃO estão corretos consultando a página de *Stack and Versions* da interface web do Ambari.

- Inicie o Ambari Server:

Terminal input

```
    ambari-server start
```

- Verifique se a inconsistência foi resolvida acessando a interface web do Ambari, clicando na opção *Stack and Versions* do menu lateral, e depois na guia *VERSIONS*.

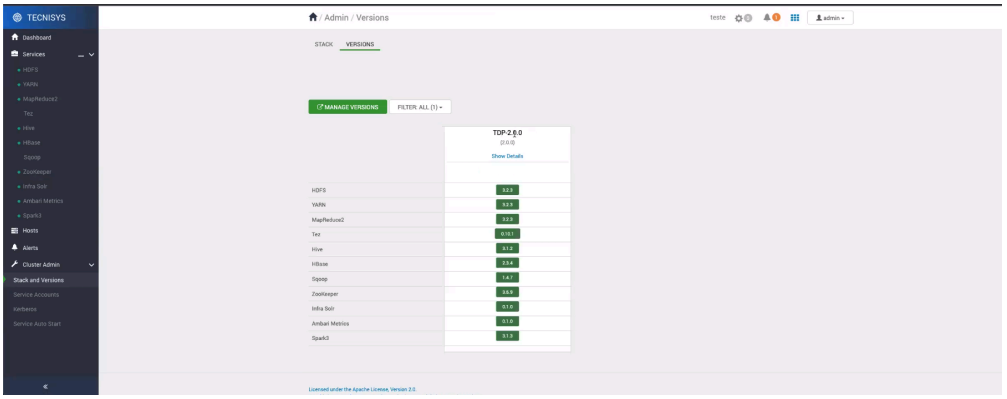


Figura 7 - Inconsistência resolvida

Registo da Nova Versão

A seguir, são apresentados os procedimentos para registar 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".

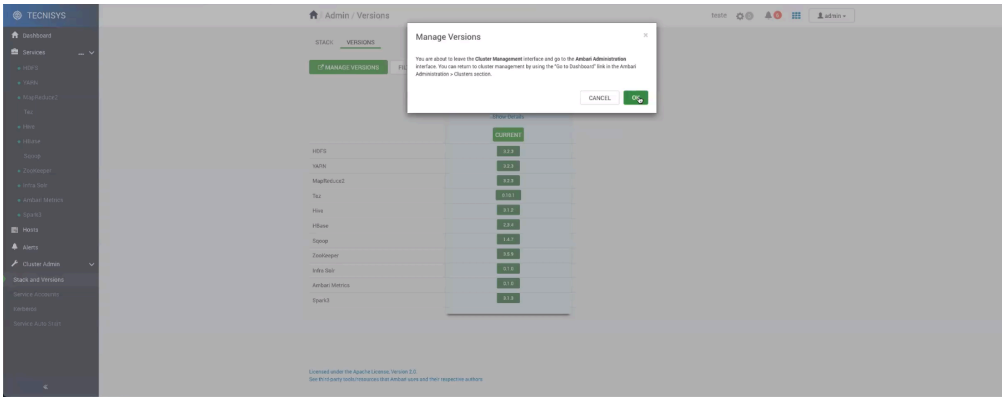


Figura 1 - Redirecionamento para a área de administração de versões

- Confirme o redirecionamento para a área de administração de versões do Ambari.

- Selecione "Register Version".

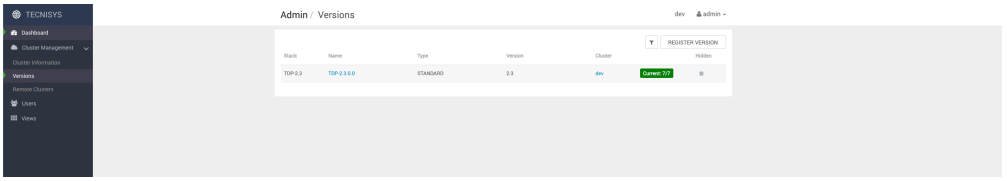


Figura 2 - Área de administração de versões

- No registo da nova versão, indique a *Base URL* dos repositórios de componentes (TDP-3.0) e utilitários (TDP-UTILS-3.0).

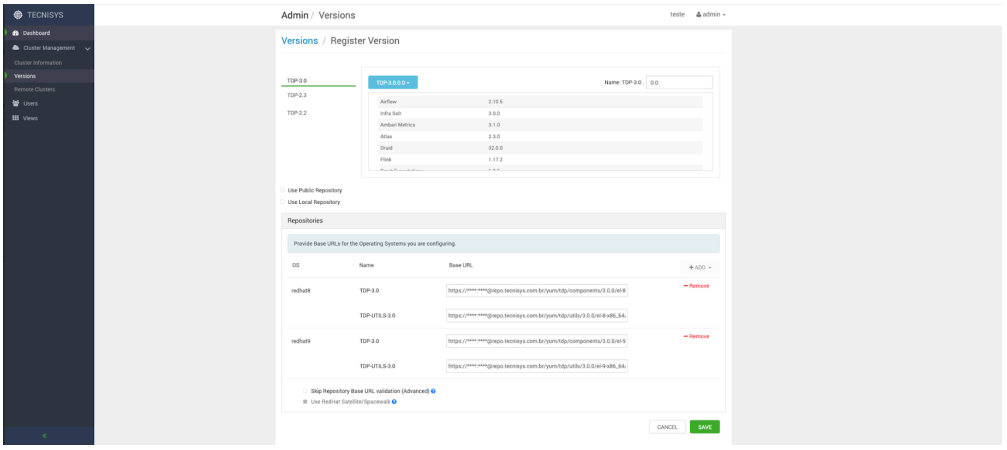


Figura 3 - Registo da nova versão

- Salve as alterações.

- De seguida, na guia *VERSIONS* já estará disponível a nova versão da *stack* do TDP.

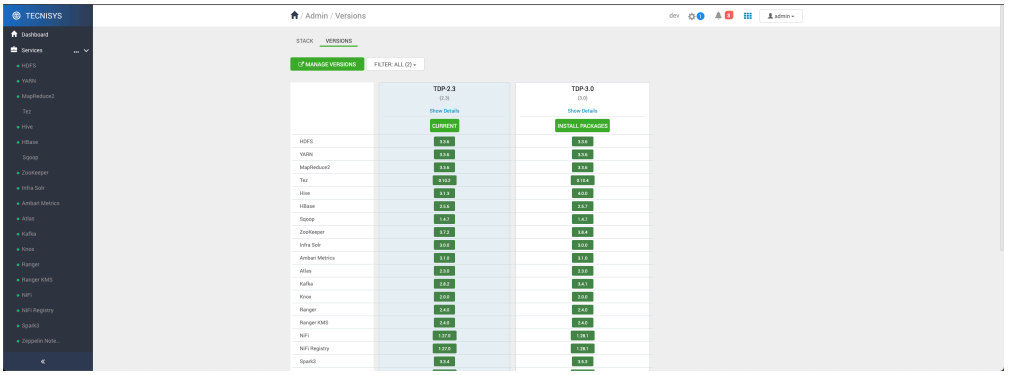


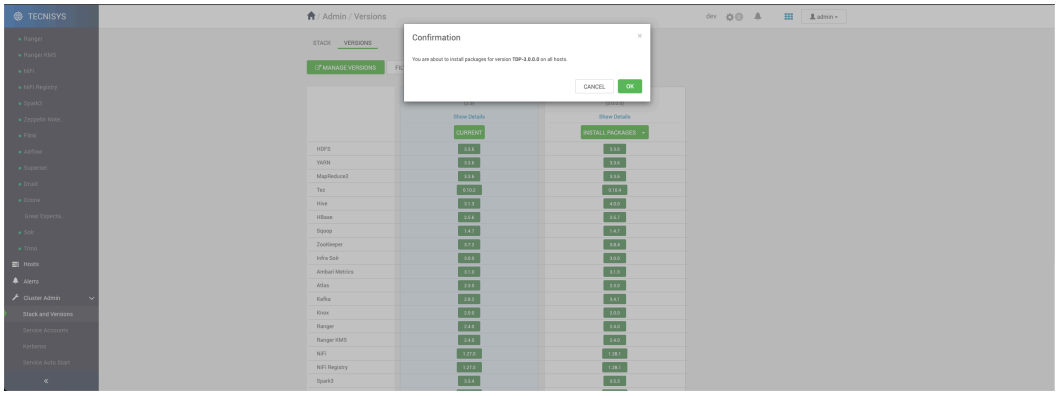
Figura 4 - Nova versão disponível

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*.

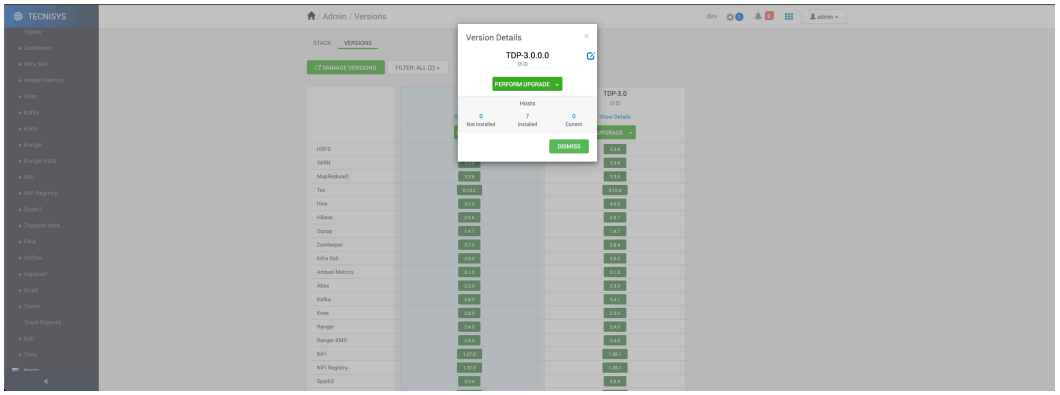
Instalação dos Pacotes

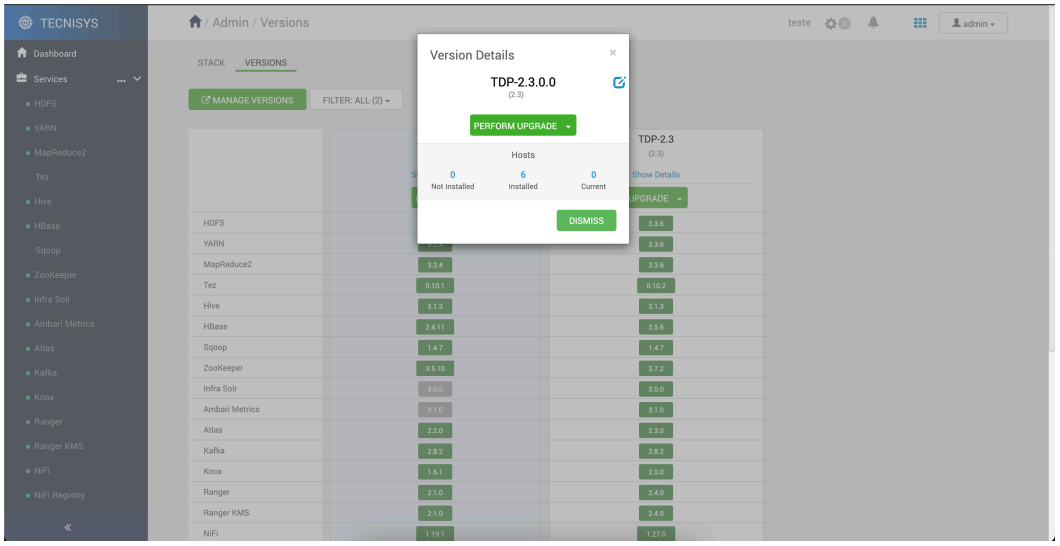
1. No Ambari, na página "Stack and Versions", guia VERSIONS, clique no botão INSTALL PACKAGES da nova versão.



2. Confirme a instalação dos pacotes.

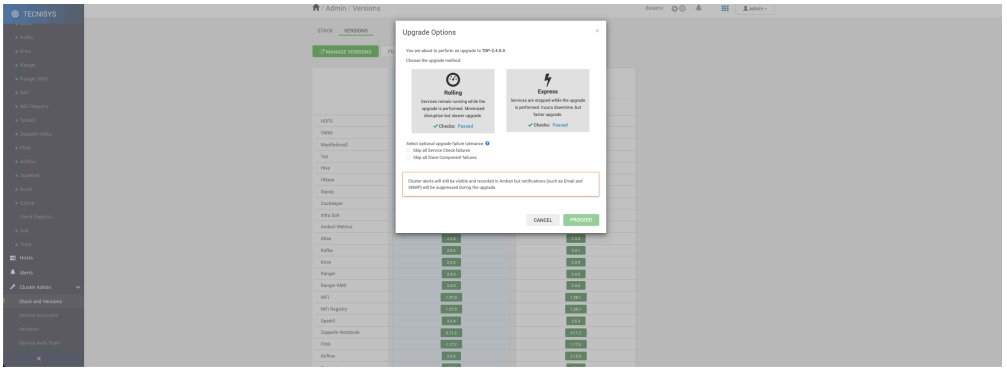
3. Clique no link "Show details" de ambas as versões para visualizar o total de máquinas com pacotes instalados.





> **NOTA**
>
> Até este momento, mesmo que os pacotes da nova versão já estejam instalados nas máquinas, o *Cluster* ainda opera com os binários dos 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](pre-requisitos-atualizacao#início-automático- dos-serviços-desativados).

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*.

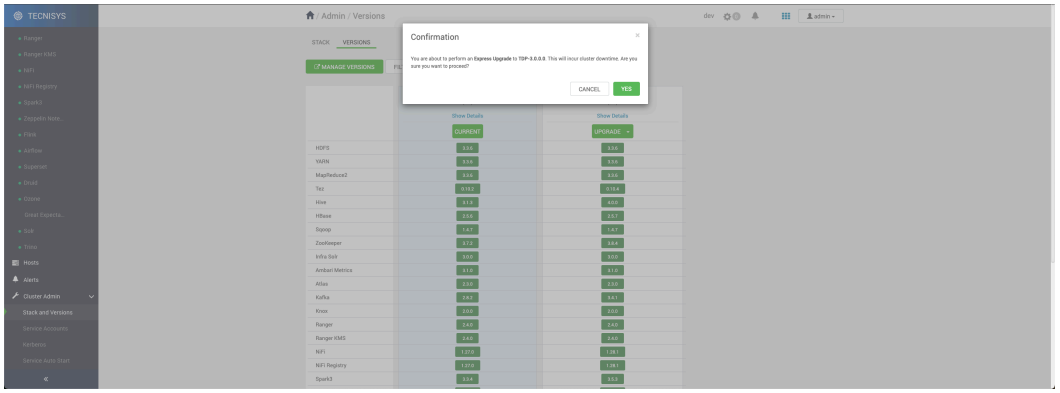
Estando todos os pré-requisitos atendidos, siga os passos a seguir:

1. Clique na botão UPGRADE da nova versão.

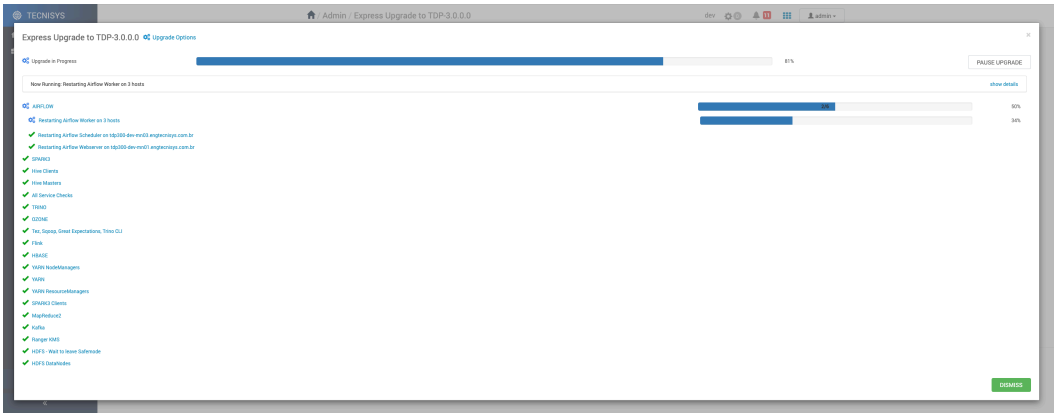
2. Selecione a opção de atualização "Express".

3. Clique em "Proceed" para continuar.

4. Confirme a Operação.

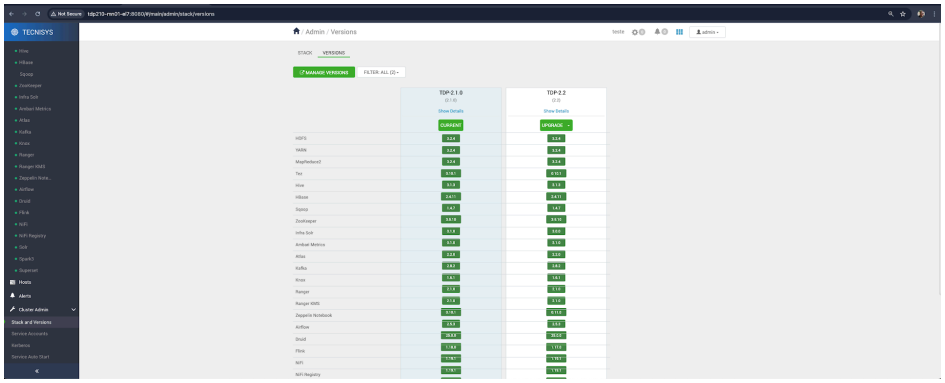


] 5. Acompanhe o processo de atualização e fique atento às mensagens e solicitações exibidas na tela.



AVISO

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.



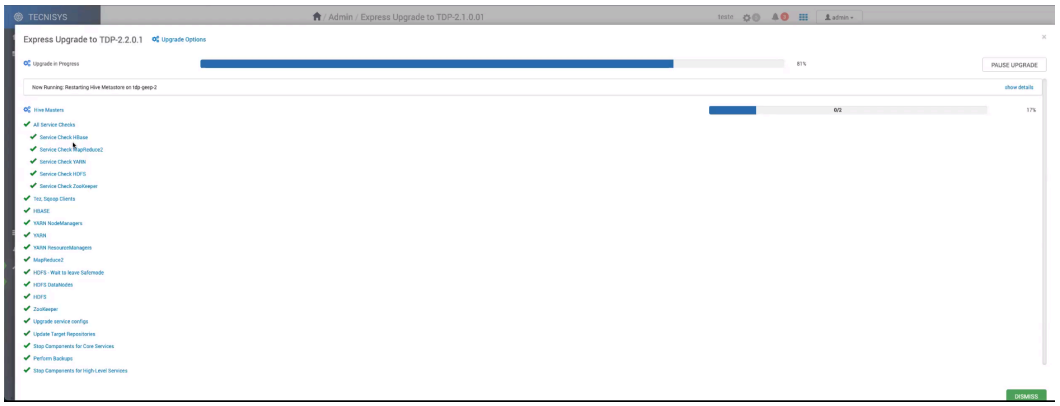
AVISO

Caso o serviço do Apache Hive esteja instalado, durante a atualização, será solicitado a realização dum *backup* da Base de Dados do Hive Metastore. Esta ação é necessária para viabilizar a recuperação em caso de falha ou o *downgrade* da versão. De seguida, de volta ao Ambari, confirme a realização do *backup* para prosseguir com a atualização.

DICA

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:

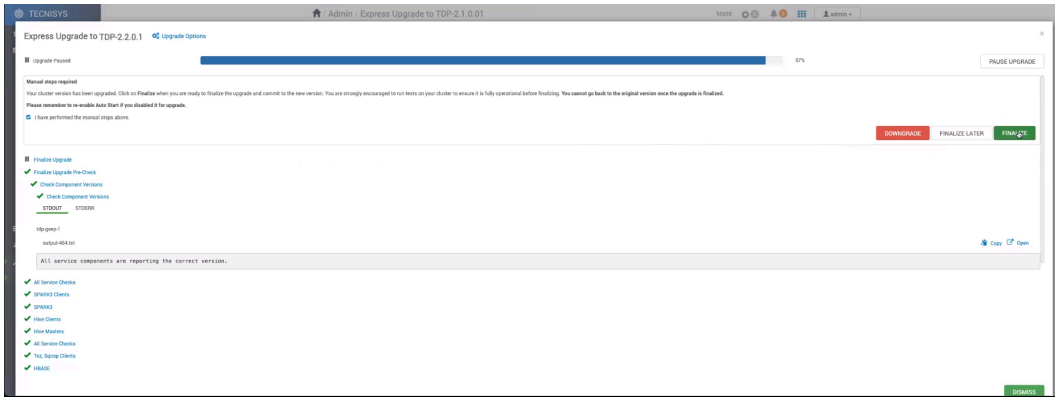
- *DOWNGRADE*: Permite reverter o processo de atualização. Todos os componentes voltam para suas versões anteriores.
- *FINALIZE LATER*: Permite finalizar o processo de atualização em outro momento, possibilitando a execução de testes e validações.
- *FINALIZE*: Finaliza o processo de atualização e define a nova versão como "CURRENT".

AVISO

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

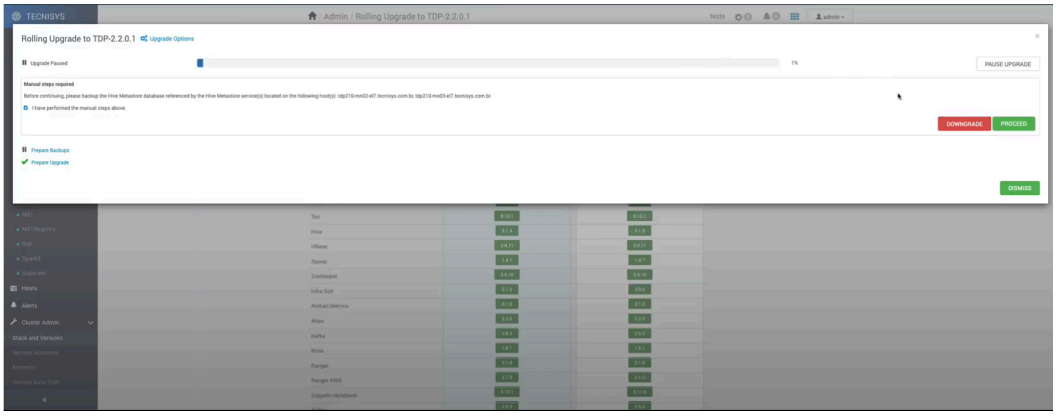
Embora seja uma opção mais demorada, o *Rolling Upgrade* permite a atualização dos componentes do *Cluster* para uma nova versão sem interromper completamente os serviços. Isso é possível desde que os serviços estejam em alta disponibilidade ou que seus componentes operem numa arquitetura distribuída em que não há um único ponto de coordenação e controlo (*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".

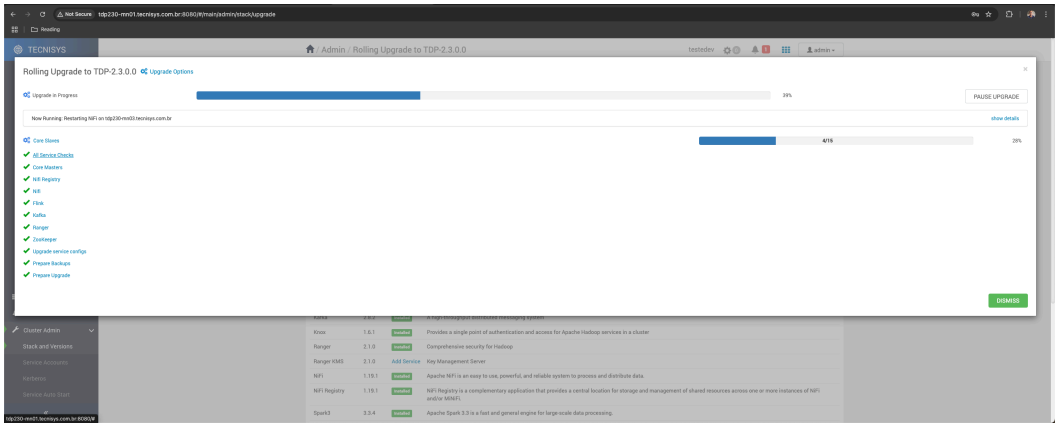
3. Confirme a realização do backup dos bancos de metadados.



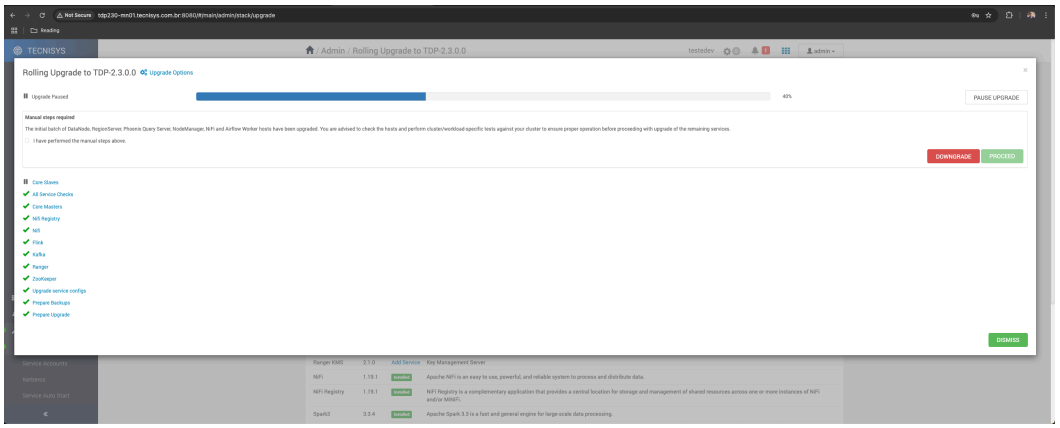
> **NOTA**

Uma série de configurações serão realizadas de forma automatizada durante a atualização.

> **NOTA**
>
> 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.



> **DICA**
>
> 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.

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".

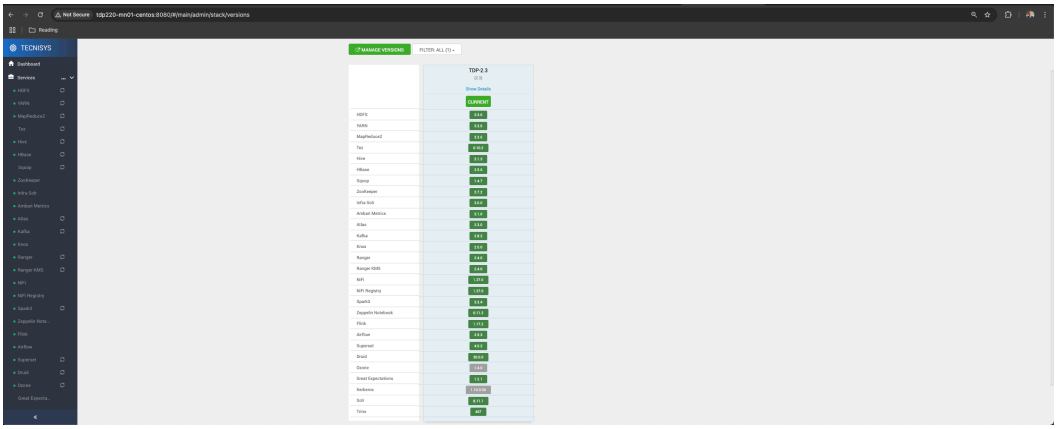
    > **AVISO**
    >
    > Caso o Apache HBase esteja entre os serviços atualizados, ao final do *upgrade* será solicitada a confirmação da exclusão do *snapshot* criado durante o processo.

7. Selecione "Finalize" para finalizar o processo de atualização.

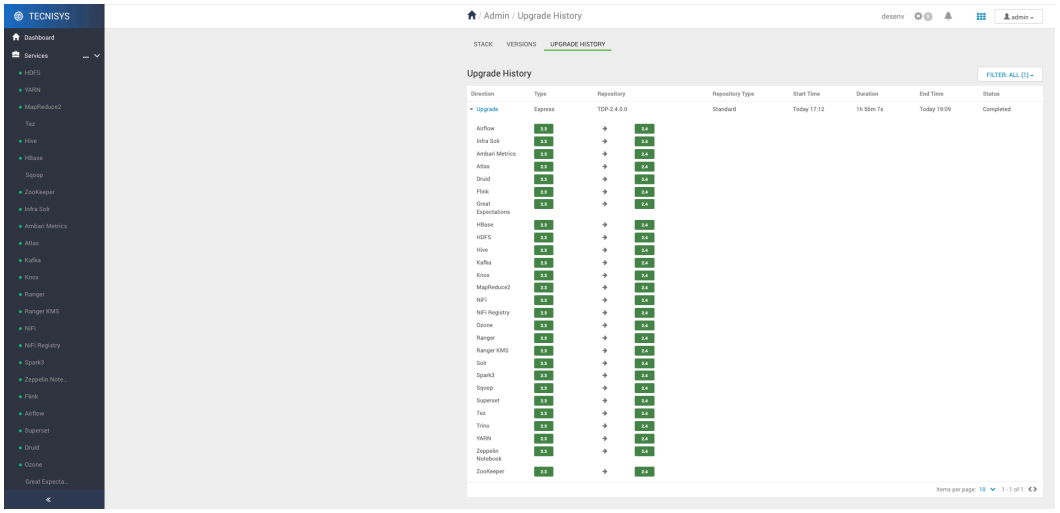
8. Clique em "Proceed" para confirmar a finalização do processo de atualização.

Conclusão da Atualização

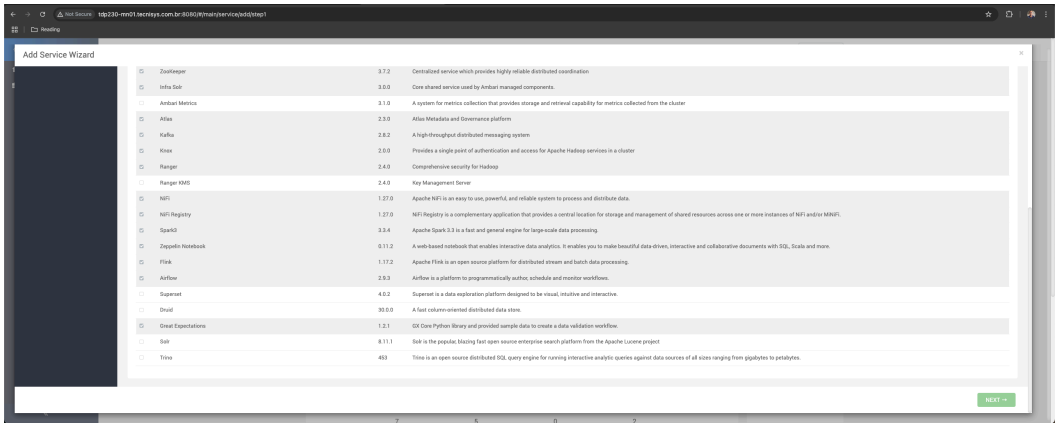
Na guia VERSIONS da página "Stack and Versions" a versão anterior foi removida e a nova versão está definida como "CURRENT".



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

A reinicialização do Ambari Server é necessária para a consolidação de determinadas alterações e configurações.

AVISO

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".

PARTE V - COMPONENTES

Matriz de Versões

Matriz oficial dos componentes disponíveis no TDP 2.2:

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Airflow	**2.5.3**
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ambari	**3.0.0.0**
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Atlas	**2.2.0**
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Druid	**25.0.0**
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	Apache Hadoop	**3.2.4**
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache HBase	**2.4.11**

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Hive	3.1.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Iceberg	1.5.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Kafka	2.8.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Knox	1.6.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Livy	0.7.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi	1.19.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi Registry	1.19.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Phoenix	5.1.3

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ranger	2.1.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Ranger KMS	2.1.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Solr	8.11.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Spark	3.3.4
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Sqoop	1.4.7
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Superset	2.1.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Tez	0.10.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zeppelin	0.11.0

Sistema Operacional	Versão SO	Arqui-tetura	Componente	Versão Componente
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zookeeper	3.5.10
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Delta Lake	2.3.0

Consulte os requisítos mínimos para a instalação do TDP aqui.

PARTE VI - NOTAS TÉCNICAS

Destaques da Versão

Plataforma	Versão	Destaques
Tecnisys Data Platform	TDP 2.2	- Suporte à sistemas operativos Enterprise Linux 8 e 9 (Rocky Linux, AlmaLinux, Red Hat, entre outros). - Adição do Apache Ozone para armazenamento escalável e flexível de objetos.. - Adição do Great Expectations (Core) para análise da qualidade de dados. - Adição do serviço do HBase Rest API. Inclusão de novos módulos para o Apache Airflow. Adição do plugin do Debezium para CDC (Change Data Capture) do PostgreSQL. Atualização dos componentes Airflow, Ambari, Atlas, Druid, Hadoop, HBase, Knox, Livy, NiFi, Phoenix, Ranger, Ranger KMS, Superset, Tez, Zeppelin e ZooKeeper. - Correção do jaas_conf do Spark para ambientes Kerberizados.

Plataforma	Versão	Destaques
		- Adicionada a propriedade hive.server2.leader.zookeeper.namespace ao Hive. - Inclusão da opção de Rolling Upgrade para novas versões da stack TDP. - Melhorias na integração de diversos componentes com SSL/TLS ativado. - Melhorias na integração dos componentes Flink, Iceberg e Zeppelin. - Adição do ResourceManager UI na topologia do Knox. - Integração do Flink com Avro, JDBC, ORC, Parquet, CSV, Hive, Kafka e Iceberg. - Integração do Spark 3.3.4 com Delta 2, Elastic, Iceberg, Kafka, PostgreSQL, Phoenix (HBase) e Solr. - Melhorias nos grafos do Ambari Metrics. - Atualização dos componentes Delta, Iceberg, Grafana (integrado ao Ambari Metrics), Phoenix e Zeppelin.

Tabela A. Destaques da Plataforma

Destaques dos Componentes

Veja também os destaques dos componentes atualizados:

Serviço	Versão	Categoria	Destaques
Apache Ambari	Ambari 3.0.0.0	Administração	- Modernização da interface, tornando-a mais intuitiva e moderna. - Melhorias e correções de problemas no Ambari Metrics. - Operação com Python 3 em sistemas operativos EL 8 e 9.
Apache Iceberg	Iceberg 1.5.0	Analytics	- Solucionado problema no FileIO no qual uma requisição extra de cabeçalho era feita ao ler manifests. - Marcados como retryable os códigos de status HTTP 502 e 504 no REST Client. - Correções e melhorias no JDBC Catalog.
Apache Phoenix	Phoenix 5.1.3	Analytics	- Melhorias na estabilidade e correções de proble-

Serviço	Versão	Categoria	Destaques
			mas conhecidos com o cliente JDBC. - Correções relacionadas à gestão de transações para garantir maior consistência e integridade dos dados. - Melhoria na execução de consultas, especialmente para cenários de grande volume de dados, reduzindo o tempo de resposta e otimizando o uso de recursos. - Melhorias na manipulação e uso de índices secundários para acelerar as operações de leitura. - Atualizações para suportar melhor a integração com o Apache Spark, permitindo o uso de Phoenix como camada de consulta SQL para dados em HBase dentro de pipeli-

Serviço	Versão	Categoria	Destaques
			nes Spark. - Maior compatibilidade com versões recentes do HBase.
Apache Spark	Spark 3.3.4	Analytics, Ciência de Dados, Graph, Streaming	- Ajustes no comportamento de diversas funções SQL, como percentile_disc e percentile_approx(), para lidar corretamente com valores NULL. - Melhoria no comportamento das funções de SQL, incluindo to_number, try_to_number e tratamento de colunas com GROUPING SETS. - Correções no comportamento de pushdown de expressões em DataSource V2 e em funções de comparação binária. - Melhoria no gerenciamento de threads e na integridade estrutural

Serviço	Versão	Categoria	Destaques
			do CoarseGrainedExecutorBackend. - Correções em problemas de desempenho relacionados ao TransportClientFactory e ao uso de await(). - Ajustes para melhorar a compatibilidade com ambientes Kubernetes e YARN, garantindo que a alocação de recursos e a renovação de tokens sejam manejadas corretamente. - Atualizações nas dependências, incluindo a versão do ORC para 1.7.10.
Apache Zeppelin	Zeppelin 0.11.0	Notebook	- Suporte à JDK 11 por padrão, o que melhora a compatibilidade e o desempenho em relação às versões anteriores. - Suporte para as versões mais re-

Serviço	Versão	Categoria	Destaques
			centes do Apache Spark (3.5.0) e Apache Flink (1.17). - Python 3.9 é agora a versão padrão para o interpretador Python, garantindo melhor suporte e desempenho para scripts Python. - Criação dinâmica de formulários dentro dos notebooks para facilitar a interação do usuário com os dados. - Melhorias nas funcionalidades de visualização de dados, incluindo grafos pivot e outras ferramentas interativas para análise de dados. - Resoluções de bugs e melhorias na segurança.

Tabela B. Destaques dos componentes

Componentes Adicionados

Os seguintes componentes foram adicionados no TDP 2.2

- Kafka-UI

Componentes Removidos

Os seguintes componentes foram removidos no TDP 2.2

- Apache Oozie

CORREÇÕES

Fix 202408001

Descrição da Inconsistência ou Falha

A atualização do navegador Chrome, versão 127.0.6533.88/89, realizada em 30/07/2024, descontinuou um recurso de apresentação utilizado pelo componente Ambari Web. Como resultado, observámos uma quebra no layout das páginas da interface web do Ambari quando acedidas através deste navegador.

Serviços e Componentes Afetados

- Ambari 2.7.6.3: Ambari Web
- Ambari 3.0.0.0: Ambari Web

Issues Relacionadas

- AMBARI-7883
- AMBARI-7884

Impacto

O conteúdo central das páginas da interface web do Ambari é apresentado abaixo do menu lateral, em vez de ao lado.

> **INFORMAÇÃO**
>
> A operação dos componentes e as funcionalidades disponíveis nas páginas do Ambari Web não foram comprometidas.

Ações Requeridas

Para corrigir esta falha, aplique a correção correspondente à versão do Apache Ambari afetada, conforme instruções a seguir.

Instruções

Apache Ambari 2.7.6.3

Esta correção resolve o problema de quebra do layout da interface web do Apache Ambari 2.7.6.3 (Enterprise Linux 7):

Execute os passos a seguir na máquina do Ambari Server para aplicar a correção:

1. **Download**

   Utilize `wget` ou `curl` para transferir o ficheiro RPM da correção a partir do repositório público de pacotes da Tecnisys.

   - Exemplo com `wget`:

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/2.7.6.3/202408/2.7.6.3-
202408001/ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

   - Exemplo com `curl`:

Terminal input

```
curl -O -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/2.7.6.3/202408/2.7.6.3-
202408001/ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

2. **Instalação**

   Utilize `yum install` para instalar a correção transferida na máquina do Ambari Server:

Terminal input

```
sudo yum install ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

Apache Ambari 3.0.0.0

Esta correção resolve o problema de quebra do layout da interface web do Apache Ambari 3.0.0.0 (Enterprise Linux 8 e 9):

Execute os passos a seguir na máquina do Ambari Server para aplicar a correção:

1. **Download**

   Utilize `wget` ou `curl` para transferir o ficheiro RPM da correção a partir do repositório público de pacotes da Tecnisys.

   - Exemplo com `wget` para EL 9:

     Terminal input

     ```
     wget --user USUARIO --password SENHA
     https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
     202408001/ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
     ```

   - Exemplo com `curl` para EL 9:

     Terminal input

     ```
     curl -O -S --user USUARIO:SENHA
     https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
     202408001/ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
     ```

2. **Instalação**

   Utilize `dnf install` para instalar a correção transferida:

   Terminal input

   ```
   sudo dnf install ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
   ```

Vídeo com wget

Carregando...

Vídeo com curl

Carregando...

Fix 202408002

Descrição da Inconsistência ou Falha

O Apache Ambari apresenta o erro do tipo `configparser.InterpolationSyntaxError` ao tentar realizar o download de pacotes de instalação.

O problema foi causado pela interpolação de variáveis do configparser, que não lida bem com caracteres especiais em URLs, como o `%40` (representação do @) utilizado para a informação de credenciais de autenticação.

Serviços e Componentes Afetados

- Ambari 3.0.0.0: Ambari Web

Issues Relacionadas

- AMBARI-7907

Impacto

Não é possível realizar o download de pacotes de instalação do Apache Ambari quando o URL do repositório contém caracteres especiais, exigindo assim a criação de um repositório de pacotes local.

Ações Requeridas

Para corrigir esta falha, siga as instruções abaixo.

Instruções

Apache Ambari 3.0.0.0

Esta correção resolve o problema de erro configparser.InterpolationSyntaxError ao realizar o download de pacotes de instalação no Apache Ambari 3.0.0.0 (Enterprise Linux 8 e 9):

Execute os passos a seguir em todas as máquinas com o Ambari Agent instalado para aplicar a correção:

1. **Download**

   Utilize `wget` ou `curl` para transferir o ficheiro RPM da correção a partir do repositório público de pacotes da Tecnisys.

   - Exemplo com `wget` para EL 9:

     Terminal input

     ```
     wget --user USUARIO --password SENHA
     https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
     202408002/ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
     ```

   - Exemplo com `curl` para EL 9:

     Terminal input

     ```
     curl -O -S --user USUARIO:SENHA
     https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
     202408002/ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
     ```

2. **Instalação**

   Utilize `dnf install` para instalar a correção transferida:

   Terminal input

   ```
   sudo dnf install ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
   ```

Vídeo com wget

Carregando...

Vídeo com curl

Carregando...

Mais Informações sobre cada componente da Plataforma TDP

A seguir disponibilizados mais informações sobre as implementações, correções e melhorias realizadas no TDP e seus componentes:

Tecnisys Data Platform 2.2

Seguem as *issues* resolvidas nesta versão:

Correções:

- STACK-5599 Hive não inicia automaticamente após mudança do Active Namenode
- STACK-5609 Problemas com a versão do Scala utilizada pelo interpretador do Spark no Zeppelin
- AMBARI-6929 Correção de problemas na coleta e exibição de métricas em sistemas operativos EL 8 e 9
- BIGTOP-7402 Pydruid não conecta com Kerberos via Superset
- STACK-7415 Erro ao habilitar SSL no Atlas e Ranger
- AMBARI-7423 Correção de problemas de configuração para uso da OpenJDK pelo Ambari
- STACK-7812 Spark não gera o `jaas_conf` em ambiente Kerberizado.

Melhorias

- AMBARI-5998 Ambari atualizado para a versão 3.0.0.0
- STACK-7249 Adicionada a propriedade `hive.server2.leader.zookeeper.namespace` ao Hive
- BIGTOP-7376 Atualização do Spark
- STACK-7429 Job Schedule habilitado por padrão no Airflow
- STACK-7430 Enriquecimento da linhagem de dados do Atlas com metadados do Airflow
- STACK-7434 Suporte a sistemas operativos Enterprise Linux 8 e 9
- BIGTOP-7559 Melhorias na integração dos componentes Flink, Iceberg e Zeppelin
- BIGTOP-7623 Atualização dos componentes Delta, Iceberg, Grafana, Phoenix e Zeppelin
- STACK-7704 Melhorias na configuração de autenticação do Hive LLAP

Novos Componentes, Recursos e Funcionalidades

- STACK-7623 Inclusão da opção de *Rolling Upgrade* no Ambari
- STACK-7390 Kafka-UI como interface gráfica para gerenciamento de clusters Kafka
- STACK-7834 Adição do HBase Rest API ao Ambari

Apache Airflow

Versão: 2.5.3

- Notas Técnicas
- GitHub

Apache Ambari

Versão: 3.0.0.0

- Notas Técnicas
- Jira
- GitHub

Apache Atlas

Versão: 2.2.0

- Notas Técnicas
- Jira
- GitHub

Delta-Lake

Versão: 2.3.0

- Notas Técnicas
- Github

Apache Druid

Versão: 25.0.0

- Notas Técnicas
- GitHub

Apache Flink

Versão: 1.17.2

- Notas Técnicas
- Jira
- GitHub

Apache Hadoop

Versão: 3.2.4

- Notas Técnicas
- Jira do Hadoop HDFS
- Jira do Hadoop YARN
- Jira do Hadoop MapReduce
- Jira do Hadoop Common
- GitHub

Apache HBase

Versão: 2.4.11

- Notas Técnicas
- Jira
- GitHub

Apache Hive

Versão: 3.1.3

- Notas Técnicas

- Jira
- GitHub

Apache Iceberg

Versão: 1.5.0

- Notas Técnicas
- GitHub

Apache Kafka

Versão: 2.8.2

- Notas Técnicas
- Jira
- GitHub

Apache Knox

Versão: 1.6.1

- Notas Técnicas
- Jira
- GitHub

Apache NiFi

Versão: 1.19.1

- Notas Técnicas
- Jira
- GitHub

Apache Phoenix

Versão: 5.1.3

- Notas Técnicas
- Jira
- GitHub

Apache Ranger

Versão: 2.1.0

- Notas Técnicas
- Jira
- GitHub

Apache Solr

Versão: 8.11.1

- Notas Técnicas
- Jira
- GitHub

Apache Spark

Versão: 3.3.4

- Notas Técnicas
- Jira
- GitHub

Apache Sqoop

Versão: 1.4.7

- Notas Técnicas
- Jira
- GitHub

Apache Superset

Versão: 2.1.3

- Notas Técnicas
- Jira
- GitHub

Apache Tez

Versão: 0.10.1

- Notas Técnicas
- Jira
- GitHub

Apache Zeppelin

Versão: 0.11.0

- Notas Técnicas
- Jira

- GitHub

Apache Zookeeper

Versão: 3.5.10

- Notas Técnicas
- Jira
- GitHub

PARTE VII - ROADMAP

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 gestão unificada de metadados.
- **Adição do ClickHouse [1]:** Serviço de base 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 gestão 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 base 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.

TDP 4.0 | 2027-S2

- **Tecnisys Data eXperience:** Camada *low code* para a construção de projetos de BI, Data Science e AI.
- **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 monitorização e gestão de clusters Kafka.
- **Suporte alargado para sistemas operativos:** 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 gestão de chaves de encriptação.
- **Atualização de componentes:** Versões mais recentes de todos os componentes.

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.
- **Atualização 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 monitorização integradas e centralizadas.
- **Repositório de pacotes exclusivo e dependências diretas:** Repositório Público de Pacotes da Tecnisys.
- **Compatibilidade com sistemas operativos:** CentOS/RHEL 7.5

Documentação v3.0

Footnotes

1. Exclusivo para a edição TDP Kubernetes. ↩

PARTE VIII - OUTRAS INFORMAÇÕES

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 aceder a nossa página de suporte.

Se o seu negócio exige atendimento prioritário, garantias (SLA), disponibilidade técnica (24x7), atendimento inicial em até 15 minutos, monitorização 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 pode abrir um chamado para relatar 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 da base de dados. Além disso, tal serviço não requer uma conexão direta com a Internet.

O ficheiro 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

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. Indique 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.

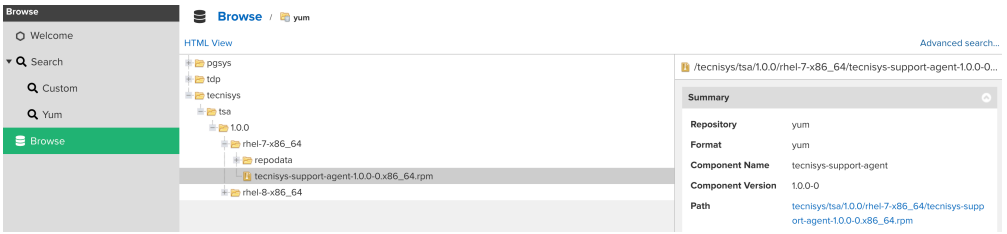


Figura 1 - Download do pacote de instalação do TSA

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
```

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 descarregar 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
```

Informações Legais

Esta documentação é mantida pela Tecnisys Informática e Assessoria Empresarial LTDA.

O TDP (Tecnisys Data Platform) é marca registada 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 ficheiros 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 contacto.

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) num 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.

Entre em contacto conosco:

E-mail: faleconosco@tecnisys.com.br Telefone: +55 61 30399700 Endereço: Trecho 8, Lote 245, SIA, Brasília, DF, Brasil, CEP 71205-080