INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

ALERTING OVERVIEW

Alerting with Prometheus is separated into two parts. Alerting rules in Prometheus servers send alerts to an Alertmanager. The Alertmanager (../alertmanager/) then manages those alerts, including silencing, inhibition, aggregation and sending out notifications via methods such as email, on-call notification systems, and chat platforms.

The main steps to setting up alerting and notifications are:

- Setup and configure (../configuration/) the Alertmanager
- Configure Prometheus (/docs/prometheus/latest/configuration/configuration/#alertmanager_config) to talk to the Alertmanager
- Create alerting rules (/docs/prometheus/latest/configuration/alerting_rules/) in Prometheus

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

ALERTMANAGER

- Grouping
- Inhibition
- Silences
- Client behavior
- High Availability

The Alertmanager (https://github.com/prometheus/alertmanager) handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts.

The following describes the core concepts the Alertmanager implements. Consult the configuration documentation (../configuration/) to learn how to use them in more detail.

Grouping

Grouping categorizes alerts of similar nature into a single notification. This is especially useful during larger outages when many systems fail at once and hundreds to thousands of alerts may be firing simultaneously.

Example: Dozens or hundreds of instances of a service are running in your cluster when a network partition occurs. Half of your service instances can no longer reach the database. Alerting rules in Prometheus were configured to send an alert for each service instance if it cannot communicate with the database. As a result hundreds of alerts are sent to Alertmanager.

As a user, one only wants to get a single page while still being able to see exactly which service instances were affected. Thus one can configure Alertmanager to group alerts by their cluster and alertname so it sends a single compact notification.

Grouping of alerts, timing for the grouped notifications, and the receivers of those notifications are configured by a routing tree in the configuration file.

Inhibition

Inhibition is a concept of suppressing notifications for certain alerts if certain other alerts are already firing.

Example: An alert is firing that informs that an entire cluster is not reachable. Alertmanager can be configured to mute all other alerts concerning this cluster if that particular alert is firing. This prevents notifications for hundreds or thousands of firing alerts that are unrelated to the actual issue.

Inhibitions are configured through the Alertmanager's configuration file.

Silences

Silences are a straightforward way to simply mute alerts for a given time. A silence is configured based on matchers, just like the routing tree. Incoming alerts are checked whether they match all the equality or regular expression matchers of an active silence. If they do, no notifications will be sent out for that alert.

Silences are configured in the web interface of the Alertmanager.

Client behavior

The Alertmanager has special requirements (../clients/) for behavior of its client. Those are only relevant for advanced use cases where Prometheus is not used to send alerts.

High Availability

Alertmanager supports configuration to create a cluster for high availability. This can be configured using the --cluster-* (https://github.com/prometheus/alertmanager#high-availability) flags.

It's important not to load balance traffic between Prometheus and its Alertmanagers, but instead, point Prometheus to a list of all Alertmanagers.

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

SENDING ALERTS

Disclaimer: Prometheus automatically takes care of sending alerts generated by its configured alerting rules (/docs/prometheus/latest/configuration/alerting_rules/). It is highly recommended to configure alerting rules in Prometheus based on time series data rather than implementing a direct client.

The Alertmanager has two APIs, v1 and v2, both listening for alerts. The scheme for v1 is described in the code snipped below. The scheme for v2 is specified as an OpenAPI specification that can be found in the Alertmanager repository (https://github.com/prometheus/alertmanager/blob/master/api/v2/openapi.yaml). Clients are expected to continuously re-send alerts as long as they are still active (usually on the order of 30 seconds to 3 minutes). Clients can push a list of alerts to Alertmanager via a POST request.

The labels of each alert are used to identify identical instances of an alert and to perform deduplication. The annotations are always set to those received most recently and are not identifying an alert.

Both `startsAt` and `endsAt` timestamp are optional. If `startsAt` is omitted, the current time is assigned by the Alertmanager. `endsAt` is only set if the end time of an alert is known. Otherwise it will be set to a configurable timeout period from the time since the alert was last received.

The `generatorURL` field is a unique back-link which identifies the causing entity of this alert in the client.

```
[
  {
    "labels": {
      "alertname": "<requiredAlertName>",
      "<labelname>": "<labelvalue>",
      ...
    },
    "annotations": {
      "<labelname>": "<labelvalue>",
    },
    "startsAt": "<rfc3339>",
    "endsAt": "<rfc3339>",
    "generatorURL": "<generator_url>"
  },
  ...
]
```

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

CONFIGURATION

- Configuration file
- `<route>`
  - Example
- `<inhibit_rule>`
- `<http_config>`
- `<tls_config>`
- `<receiver>`
- `<email_config>`
- `<hipchat_config>`
- `<pagerduty_config>`
  - `<image_config>`
  - `<link_config>`
- `<pushover_config>`
- `<slack_config>`
  - `<action_config>`
  - `<field_config>`
- `<opsgenie_config>`
  - `<responder>`
- `<victorops_config>`
- `<webhook_config>`
- `<wechat_config>`

Alertmanager (https://github.com/prometheus/alertmanager) is configured via command-line flags and a configuration file. While the command-line flags configure immutable system parameters, the configuration file defines inhibition rules, notification routing and notification receivers.

The visual editor (https://www.prometheus.io/webtools/alerting/routing-tree-editor) can assist in building routing trees.

To view all available command-line flags, run `alertmanager -h`.

Alertmanager can reload its configuration at runtime. If the new configuration is not well-formed, the changes will not be applied and an error is logged. A configuration reload is triggered by sending a `SIGHUP` to the process or sending a HTTP POST request to the `/-/reload` endpoint.

Configuration file

To specify which configuration file to load, use the `--config.file` flag.

```
./alertmanager --config.file=alertmanager.yml
```

The file is written in the YAML format (https://en.wikipedia.org/wiki/YAML), defined by the scheme described below. Brackets indicate that a parameter is optional. For non-list parameters the value is set to the specified default.

Generic placeholders are defined as follows:

- `<duration>` : a duration matching the regular expression `[0-9]+(ms|[smhdwy])`
- `<labelname>` : a string matching the regular expression `[a-zA-Z_][a-zA-Z0-9_]*`
- `<labelvalue>` : a string of unicode characters
- `<filepath>` : a valid path in the current working directory
- `<boolean>` : a boolean that can take the values `true` or `false`
- `<string>` : a regular string
- `<secret>` : a regular string that is a secret, such as a password
- `<tmpl_string>` : a string which is template-expanded before usage
- `<tmpl_secret>` : a string which is template-expanded before usage that is a secret

The other placeholders are specified separately.

A provided valid example file (https://github.com/prometheus/alertmanager/blob/master/doc/examples/simple.yml) shows usage in context.

The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.

```
global:
  # The default SMTP From header field.
  [ smtp_from: <tmpl_string> ]
  # The default SMTP smarthost used for sending emails, including port number.
  # Port number usually is 25, or 587 for SMTP over TLS (sometimes referred to as STAR
  # Example: smtp.example.org:587
  [ smtp_smarthost: <string> ]
  # The default hostname to identify to the SMTP server.
  [ smtp_hello: <string> | default = "localhost" ]
  # SMTP Auth using CRAM-MD5, LOGIN and PLAIN. If empty, Alertmanager doesn't authenti
  [ smtp_auth_username: <string> ]
  # SMTP Auth using LOGIN and PLAIN.
  [ smtp_auth_password: <secret> ]
  # SMTP Auth using PLAIN.
  [ smtp_auth_identity: <string> ]
  # SMTP Auth using CRAM-MD5.
  [ smtp_auth_secret: <secret> ]
  # The default SMTP TLS requirement.
  # Note that Go does not support unencrypted connections to remote SMTP endpoints.
  [ smtp_require_tls: <bool> | default = true ]

  # The API URL to use for Slack notifications.
  [ slack_api_url: <secret> ]
  [ victorops_api_key: <secret> ]
  [ victorops_api_url: <string> | default = "https://alert.victorops.com/integrations/
  [ pagerduty_url: <string> | default = "https://events.pagerduty.com/v2/enqueue" ]
  [ opsgenie_api_key: <secret> ]
  [ opsgenie_api_url: <string> | default = "https://api.opsgenie.com/" ]
  [ hipchat_api_url: <string> | default = "https://api.hipchat.com/" ]
  [ hipchat_auth_token: <secret> ]
  [ wechat_api_url: <string> | default = "https://qyapi.weixin.qq.com/cgi-bin/" ]
  [ wechat_api_secret: <secret> ]
  [ wechat_api_corp_id: <string> ]

  # The default HTTP client configuration
  [ http_config: <http_config> ]

  # ResolveTimeout is the default value used by alertmanager if the alert does
  # not include EndsAt, after this time passes it can declare the alert as resolved if
  # This has no impact on alerts from Prometheus, as they always include EndsAt.
  [ resolve_timeout: <duration> | default = 5m ]

# Files from which custom notification template definitions are read.
# The last component may use a wildcard matcher, e.g. 'templates/*.tmpl'.
templates:
  [ - <filepath> ... ]
```

```
# The root node of the routing tree.
route: <route>

# A list of notification receivers.
receivers:
  - <receiver> ...

# A list of inhibition rules.
inhibit_rules:
  [ - <inhibit_rule> ... ]
```

`<route>`

A route block defines a node in a routing tree and its children. Its optional configuration parameters are inherited from its parent node if not set.

Every alert enters the routing tree at the configured top-level route, which must match all alerts (i.e. not have any configured matchers). It then traverses the child nodes. If `continue` is set to false, it stops after the first matching child. If `continue` is true on a matching node, the alert will continue matching against subsequent siblings. If an alert does not match any children of a node (no matching child nodes, or none exist), the alert is handled based on the configuration parameters of the current node.

```
[ receiver: <string> ]
# The labels by which incoming alerts are grouped together. For example,
# multiple alerts coming in for cluster=A and alertname=LatencyHigh would
# be batched into a single group.
#
# To aggregate by all possible labels use the special value '...' as the sole label na
# group_by: ['...']
# This effectively disables aggregation entirely, passing through all
# alerts as-is. This is unlikely to be what you want, unless you have
# a very low alert volume or your upstream notification system performs
# its own grouping.
[ group_by: '[' <labelname>, ... ']' ]

# Whether an alert should continue matching subsequent sibling nodes.
[ continue: <boolean> | default = false ]

# A set of equality matchers an alert has to fulfill to match the node.
match:
  [ <labelname>: <labelvalue>, ... ]

# A set of regex-matchers an alert has to fulfill to match the node.
match_re:
  [ <labelname>: <regex>, ... ]

# How long to initially wait to send a notification for a group
# of alerts. Allows to wait for an inhibiting alert to arrive or collect
# more initial alerts for the same group. (Usually ~0s to few minutes.)
[ group_wait: <duration> | default = 30s ]

# How long to wait before sending a notification about new alerts that
# are added to a group of alerts for which an initial notification has
# already been sent. (Usually ~5m or more.)
[ group_interval: <duration> | default = 5m ]

# How long to wait before sending a notification again if it has already
# been sent successfully for an alert. (Usually ~3h or more).
[ repeat_interval: <duration> | default = 4h ]

# Zero or more child routes.
routes:
  [ - <route> ... ]
```

Example

```
# The root route with all parameters, which are inherited by the child
# routes if they are not overwritten.
route:
  receiver: 'default-receiver'
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  group_by: [cluster, alertname]
  # All alerts that do not match the following child routes
  # will remain at the root node and be dispatched to 'default-receiver'.
  routes:
  # All alerts with service=mysql or service=cassandra
  # are dispatched to the database pager.
  - receiver: 'database-pager'
    group_wait: 10s
    match_re:
      service: mysql|cassandra
  # All alerts with the team=frontend label match this sub-route.
  # They are grouped by product and environment rather than cluster
  # and alertname.
  - receiver: 'frontend-pager'
    group_by: [product, environment]
    match:
      team: frontend
```

`<inhibit_rule>`

An inhibition rule mutes an alert (target) matching a set of matchers when an alert (source) exists that matches another set of matchers. Both target and source alerts must have the same label values for the label names in the `equal` list.

Semantically, a missing label and a label with an empty value are the same thing. Therefore, if all the label names listed in `equal` are missing from both the source and target alerts, the inhibition rule will apply.

To prevent an alert from inhibiting itself, an alert that matches *both* the target and the source side of a rule cannot be inhibited by alerts for which the same is true (including itself). However, we recommend to choose target and source matchers in a way that alerts never match both sides. It is much easier to reason about and does not trigger this special case.

```
# Matchers that have to be fulfilled in the alerts to be muted.
target_match:
  [ <labelname>: <labelvalue>, ... ]
target_match_re:
  [ <labelname>: <regex>, ... ]

# Matchers for which one or more alerts have to exist for the
# inhibition to take effect.
source_match:
  [ <labelname>: <labelvalue>, ... ]
source_match_re:
  [ <labelname>: <regex>, ... ]

# Labels that must have an equal value in the source and target
# alert for the inhibition to take effect.
[ equal: '[' <labelname>, ... ']' ]
```

`<http_config>`

A `http_config` allows configuring the HTTP client that the receiver uses to communicate with HTTP-based API services.

```
# Note that `basic_auth`, `bearer_token` and `bearer_token_file` options are
# mutually exclusive.

# Sets the `Authorization` header with the configured username and password.
# password and password_file are mutually exclusive.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Sets the `Authorization` header with the configured bearer token.
[ bearer_token: <secret> ]

# Sets the `Authorization` header with the bearer token read from the configured file.
[ bearer_token_file: <filepath> ]

# Configures the TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]
```

`<tls_config>`

A `tls_config` allows configuring TLS connections.

```
# CA certificate to validate the server certificate with.
[ ca_file: <filepath> ]

# Certificate and key files for client cert authentication to the server.
[ cert_file: <filepath> ]
[ key_file: <filepath> ]

# ServerName extension to indicate the name of the server.
# http://tools.ietf.org/html/rfc4366#section-3.1
[ server_name: <string> ]

# Disable validation of the server certificate.
[ insecure_skip_verify: <boolean> | default = false]
```

`<receiver>`

Receiver is a named configuration of one or more notification integrations.

We're not actively adding new receivers, we recommend implementing custom notification integrations via the webhook receiver.

```
# The unique name of the receiver.
name: <string>

# Configurations for several notification integrations.
email_configs:
  [ - <email_config>, ... ]
hipchat_configs:
  [ - <hipchat_config>, ... ]
pagerduty_configs:
  [ - <pagerduty_config>, ... ]
pushover_configs:
  [ - <pushover_config>, ... ]
slack_configs:
  [ - <slack_config>, ... ]
opsgenie_configs:
  [ - <opsgenie_config>, ... ]
webhook_configs:
  [ - <webhook_config>, ... ]
victorops_configs:
  [ - <victorops_config>, ... ]
wechat_configs:
  [ - <wechat_config>, ... ]
```

`<email_config>`

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

# The email address to send notifications to.
to: <tmpl_string>

# The sender address.
[ from: <tmpl_string> | default = global.smtp_from ]

# The SMTP host through which emails are sent.
[ smarthost: <string> | default = global.smtp_smarthost ]

# The hostname to identify to the SMTP server.
[ hello: <string> | default = global.smtp_hello ]

# SMTP authentication information.
[ auth_username: <string> | default = global.smtp_auth_username ]
[ auth_password: <secret> | default = global.smtp_auth_password ]
[ auth_secret: <secret> | default = global.smtp_auth_secret ]
[ auth_identity: <string> | default = global.smtp_auth_identity ]

# The SMTP TLS requirement.
# Note that Go does not support unencrypted connections to remote SMTP endpoints.
[ require_tls: <bool> | default = global.smtp_require_tls ]

# TLS configuration.
tls_config:
  [ <tls_config> ]

# The HTML body of the email notification.
[ html: <tmpl_string> | default = '{{ template "email.default.html" . }}' ]
# The text body of the email notification.
[ text: <tmpl_string> ]

# Further headers email header key/value pairs. Overrides any headers
# previously set by the notification implementation.
[ headers: { <string>: <tmpl_string>, ... } ]
```

`<hipchat_config>`

HipChat notifications use a Build Your Own (https://confluence.atlassian.com/hc/integrations-with-hipchat-server-683508267.html) integration.

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

# The HipChat Room ID.
room_id: <tmpl_string>
# The auth token.
[ auth_token: <secret> | default = global.hipchat_auth_token ]
# The URL to send API requests to.
[ api_url: <string> | default = global.hipchat_api_url ]

# See https://www.hipchat.com/docs/apiv2/method/send_room_notification
# A label to be shown in addition to the sender's name.
[ from:  <tmpl_string> | default = '{{ template "hipchat.default.from" . }}' ]
# The message body.
[ message:  <tmpl_string> | default = '{{ template "hipchat.default.message" . }}' ]
# Whether this message should trigger a user notification.
[ notify:  <boolean> | default = false ]
# Determines how the message is treated by the alertmanager and rendered inside HipCha
[ message_format:  <string> | default = 'text' ]
# Background color for message.
[ color:  <tmpl_string> | default = '{{ if eq .Status "firing" }}red{{ else }}green{{

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<pagerduty_config>`

PagerDuty notifications are sent via the PagerDuty API (https://developer.pagerduty.com/documentation/integration/events). PagerDuty provides documentation (https://www.pagerduty.com/docs/guides/prometheus-integration-guide/) on how to integrate. There are important differences with Alertmanager's v0.11 and greater support of PagerDuty's Events API v2.

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The following two options are mutually exclusive.
# The PagerDuty integration key (when using PagerDuty integration type `Events API v2`
routing_key: <tmpl_secret>
# The PagerDuty integration key (when using PagerDuty integration type `Prometheus`).
service_key: <tmpl_secret>

# The URL to send API requests to
[ url: <string> | default = global.pagerduty_url ]

# The client identification of the Alertmanager.
[ client:  <tmpl_string> | default = '{{ template "pagerduty.default.client" . }}' ]
# A backlink to the sender of the notification.
[ client_url:  <tmpl_string> | default = '{{ template "pagerduty.default.clientURL" .

# A description of the incident.
[ description: <tmpl_string> | default = '{{ template "pagerduty.default.description"

# Severity of the incident.
[ severity: <tmpl_string> | default = 'error' ]

# A set of arbitrary key/value pairs that provide further detail
# about the incident.
[ details: { <string>: <tmpl_string>, ... } | default = {
  firing:       '{{ template "pagerduty.default.instances" .Alerts.Firing }}'
  resolved:     '{{ template "pagerduty.default.instances" .Alerts.Resolved }}'
  num_firing:   '{{ .Alerts.Firing | len }}'
  num_resolved: '{{ .Alerts.Resolved | len }}'
} ]

# Images to attach to the incident.
images:
  [ <image_config> ... ]

# Links to attach to the incident.
links:
  [ <link_config> ... ]

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<image_config>`

The fields are documented in the PagerDuty API documentation (https://v2.developer.pagerduty.com/v2/docs/send-an-event-events-api-v2#section-the-images-property).

```
href: <tmpl_string>
source: <tmpl_string>
alt: <tmpl_string>
```

`<link_config>`

The fields are documented in the PagerDuty API documentation (https://v2.developer.pagerduty.com/v2/docs/send-an-event-events-api-v2#section-the-links-property).

```
href: <tmpl_string>
text: <tmpl_string>
```

`<pushover_config>`

Pushover notifications are sent via the Pushover API (https://pushover.net/api).

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The recipient user’s user key.
user_key: <secret>

# Your registered application’s API token, see https://pushover.net/apps
token: <secret>

# Notification title.
[ title: <tmpl_string> | default = '{{ template "pushover.default.title" . }}' ]

# Notification message.
[ message: <tmpl_string> | default = '{{ template "pushover.default.message" . }}' ]

# A supplementary URL shown alongside the message.
[ url: <tmpl_string> | default = '{{ template "pushover.default.url" . }}' ]

# Priority, see https://pushover.net/api#priority
[ priority: <tmpl_string> | default = '{{ if eq .Status "firing" }}2{{ else }}0{{ end

# How often the Pushover servers will send the same notification to the user.
# Must be at least 30 seconds.
[ retry: <duration> | default = 1m ]

# How long your notification will continue to be retried for, unless the user
# acknowledges the notification.
[ expire: <duration> | default = 1h ]

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<slack_config>`

Slack notifications are sent via Slack webhooks (https://api.slack.com/incoming-webhooks). The notification contains an attachment (https://api.slack.com/docs/message-attachments).

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

# The Slack webhook URL.
[ api_url: <secret> | default = global.slack_api_url ]

# The channel or user to send notifications to.
channel: <tmpl_string>

# API request data as defined by the Slack webhook API.
[ icon_emoji: <tmpl_string> ]
[ icon_url: <tmpl_string> ]
[ link_names: <boolean> | default = false ]
[ username: <tmpl_string> | default = '{{ template "slack.default.username" . }}' ]
# The following parameters define the attachment.
actions:
  [ <action_config> ... ]
[ callback_id: <tmpl_string> | default = '{{ template "slack.default.callbackid" . }}'
[ color: <tmpl_string> | default = '{{ if eq .Status "firing" }}danger{{ else }}good{{
[ fallback: <tmpl_string> | default = '{{ template "slack.default.fallback" . }}' ]
fields:
  [ <field_config> ... ]
[ footer: <tmpl_string> | default = '{{ template "slack.default.footer" . }}' ]
[ mrkdwn_in: '[' <string>, ... ']' | default = ["fallback", "pretext", "text"] ]
[ pretext: <tmpl_string> | default = '{{ template "slack.default.pretext" . }}' ]
[ short_fields: <boolean> | default = false ]
[ text: <tmpl_string> | default = '{{ template "slack.default.text" . }}' ]
[ title: <tmpl_string> | default = '{{ template "slack.default.title" . }}' ]
[ title_link: <tmpl_string> | default = '{{ template "slack.default.titlelink" . }}' ]
[ image_url: <tmpl_string> ]
[ thumb_url: <tmpl_string> ]

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<action_config>`

The fields are documented in the Slack API documentation for message attachments (https://api.slack.com/docs/message-attachments#action_fields) and interactive messages (https://api.slack.com/docs/interactive-message-field-guide#action_fields).

```
text: <tmpl_string>
type: <tmpl_string>
# Either url or name and value are mandatory.
[ url: <tmpl_string> ]
[ name: <tmpl_string> ]
[ value: <tmpl_string> ]

[ confirm: <action_confirm_field_config> ]
[ style: <tmpl_string> | default = '' ]
```

`<action_confirm_field_config>`

The fields are documented in the Slack API documentation (https://api.slack.com/docs/interactive-message-field-guide#confirmation_fields).

```
text: <tmpl_string>
[ dismiss_text: <tmpl_string> | default '' ]
[ ok_text: <tmpl_string> | default '' ]
[ title: <tmpl_string> | default '' ]
```

`<field_config>`

The fields are documented in the Slack API documentation (https://api.slack.com/docs/message-attachments#fields).

```
title: <tmpl_string>
value: <tmpl_string>
[ short: <boolean> | default = slack_config.short_fields ]
```

`<opsgenie_config>`

OpsGenie notifications are sent via the OpsGenie API (https://docs.opsgenie.com/docs/alert-api).

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The API key to use when talking to the OpsGenie API.
[ api_key: <secret> | default = global.opsgenie_api_key ]

# The host to send OpsGenie API requests to.
[ api_url: <string> | default = global.opsgenie_api_url ]

# Alert text limited to 130 characters.
[ message: <tmpl_string> ]

# A description of the incident.
[ description: <tmpl_string> | default = '{{ template "opsgenie.default.description" .

# A backlink to the sender of the notification.
[ source: <tmpl_string> | default = '{{ template "opsgenie.default.source" . }}' ]

# A set of arbitrary key/value pairs that provide further detail
# about the incident.
[ details: { <string>: <tmpl_string>, ... } ]

# List of responders responsible for notifications.
responders:
  [ - <responder> ... ]

# Comma separated list of tags attached to the notifications.
[ tags: <tmpl_string> ]

# Additional alert note.
[ note: <tmpl_string> ]

# Priority level of alert. Possible values are P1, P2, P3, P4, and P5.
[ priority: <tmpl_string> ]

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<responder>`

```
# Exactly one of these fields should be defined.
[ id: <tmpl_string> ]
[ name: <tmpl_string> ]
[ username: <tmpl_string> ]

# "team", "user", "escalation" or schedule".
type: <tmpl_string>
```

`<victorops_config>`

VictorOps notifications are sent out via the VictorOps API (https://help.victorops.com/knowledge-base/victorops-restendpoint-integration/)

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The API key to use when talking to the VictorOps API.
[ api_key: <secret> | default = global.victorops_api_key ]

# The VictorOps API URL.
[ api_url: <string> | default = global.victorops_api_url ]

# A key used to map the alert to a team.
routing_key: <tmpl_string>

# Describes the behavior of the alert (CRITICAL, WARNING, INFO).
[ message_type: <tmpl_string> | default = 'CRITICAL' ]

# Contains summary of the alerted problem.
[ entity_display_name: <tmpl_string> | default = '{{ template "victorops.default.entit

# Contains long explanation of the alerted problem.
[ state_message: <tmpl_string> | default = '{{ template "victorops.default.state_messa

# The monitoring tool the state message is from.
[ monitoring_tool: <tmpl_string> | default = '{{ template "victorops.default.monitorin

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

`<webhook_config>`

The webhook receiver allows configuring a generic receiver.

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The endpoint to send HTTP POST requests to.
url: <string>

# The HTTP client's configuration.
[ http_config: <http_config> | default = global.http_config ]
```

The Alertmanager will send HTTP POST requests in the following JSON format to the configured endpoint:

```
{
  "version": "4",
  "groupKey": <string>,    // key identifying the group of alerts (e.g. to deduplicate
  "status": "<resolved|firing>",
  "receiver": <string>,
  "groupLabels": <object>,
  "commonLabels": <object>,
  "commonAnnotations": <object>,
  "externalURL": <string>,  // backlink to the Alertmanager.
  "alerts": [
    {
      "status": "<resolved|firing>",
      "labels": <object>,
      "annotations": <object>,
      "startsAt": "<rfc3339>",
      "endsAt": "<rfc3339>",
      "generatorURL": <string> // identifies the entity that caused the alert
    },
    ...
  ]
}
```

There is a list of integrations (/docs/operating/integrations/#alertmanager-webhook-receiver) with this feature.

`<wechat_config>`

WeChat notifications are sent via the WeChat API (http://admin.wechat.com/wiki/index.php?title=Customer_Service_Messages).

```
# Whether or not to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

# The API key to use when talking to the WeChat API.
[ api_secret: <secret> | default = global.wechat_api_secret ]

# The WeChat API URL.
[ api_url: <string> | default = global.wechat_api_url ]

# The corp id for authentication.
[ corp_id: <string> | default = global.wechat_api_corp_id ]

# API request data as defined by the WeChat API.
[ message: <tmpl_string> | default = '{{ template "wechat.default.message" . }}' ]
[ agent_id: <string> | default = '{{ template "wechat.default.agent_id" . }}' ]
[ to_user: <string> | default = '{{ template "wechat.default.to_user" . }}' ]
[ to_party: <string> | default = '{{ template "wechat.default.to_party" . }}' ]
[ to_tag: <string> | default = '{{ template "wechat.default.to_tag" . }}' ]
```

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

MANAGEMENT API

- Health check
- Readiness check
- Reload

Alertmanager provides a set of management API to ease automation and integrations.

Health check

```
GET /-/healthy
```

This endpoint always returns 200 and should be used to check Alertmanager health.

Readiness check

```
GET /-/ready
```

This endpoint returns 200 when Alertmanager is ready to serve traffic (i.e. respond to queries).

Reload

```
POST /-/reload
```

This endpoint triggers a reload of the Alertmanager configuration file.

An alternative way to trigger a configuration reload is by sending a `SIGHUP` to the Alertmanager process.

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

NOTIFICATION TEMPLATE EXAMPLES

The following are all different examples of alerts and corresponding Alertmanager configuration file setups (alertmanager.yml). Each use the Go templating (https://golang.org/pkg/text/template/) system.

- Customizing Slack notifications
- Accessing annotations in CommonAnnotations
- Ranging over all received Alerts
- Defining reusable templates

Customizing Slack notifications

In this example we've customised our Slack notification to send a URL to our organisation's wiki on how to deal with the particular alert that's been sent.

```
global:
  slack_api_url: '<slack_webhook_url>'

route:
  receiver: 'slack-notifications'
  group_by: [alertname, datacenter, app]

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#alerts'
    text: 'https://internal.myorg.net/wiki/alerts/{{ .GroupLabels.app }}/{{ .Grou
```

Accessing annotations in CommonAnnotations

In this example we again customize the text sent to our Slack receiver accessing the `summary` and `description` stored in the `CommonAnnotations` of the data sent by the Alertmanager.

Alert

```
groups:
- name: Instances
  rules:
  - alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
      severity: page
    # Prometheus templates apply here in the annotation and label fields of the a
    annotations:
      description: '{{ $labels.instance }} of job {{ $labels.job }} has been dowr
      summary: 'Instance {{ $labels.instance }} down'
```

Receiver

```
- name: 'team-x'
  slack_configs:
  - channel: '#alerts'
    # Alertmanager templates apply here.
    text: "<!channel> \nsummary: {{ .CommonAnnotations.summary }}\ndescription: {
```

Ranging over all received Alerts

Finally, assuming the same alert as the previous example, we customize our receiver to range over all of the alerts received from the Alertmanager, printing their respective annotation summaries and descriptions on new lines.

Receiver

```
- name: 'default-receiver'
  slack_configs:
  - channel: '#alerts'
    title: "{{ range .Alerts }}{{ .Annotations.summary }}\n{{ end }}"
    text: "{{ range .Alerts }}{{ .Annotations.description }}\n{{ end }}"
```

Defining reusable templates

Going back to our first example, we can also provide a file containing named templates which are then loaded by Alertmanager in order to avoid complex templates that span many lines. Create a file under `/alertmanager/template/myorg.tmpl` and create a template in it named "slack.myorg.txt":

```
{{ define "slack.myorg.text" }}https://internal.myorg.net/wiki/alerts/{{ .GroupLa
```

The configuration now loads the template with the given name for the "text" field and we provide a path to our custom template file:

```
global:
  slack_api_url: '<slack_webhook_url>'

route:
  receiver: 'slack-notifications'
  group_by: [alertname, datacenter, app]

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#alerts'
    text: '{{ template "slack.myorg.text" . }}'

templates:
- '/etc/alertmanager/templates/myorg.tmpl'
```

This example is explained in further detail in this blogpost (/blog/2016/03/03/custom-alertmanager-templates/).

INTRODUCTION

CONCEPTS

PROMETHEUS SERVER

VISUALIZATION

INSTRUMENTING

OPERATING

ALERT MANAGER

Version: 0.20

Alerting overview (/docs/alerting/0.20/overview/)

Alertmanager (/docs/alerting/0.20/alertmanager/)

Configuration (/docs/alerting/0.20/configuration/)

Clients (/docs/alerting/0.20/clients/)

Notification template reference (/docs/alerting/0.20/notifications/)

Notification template examples (/docs/alerting/0.20/notification_examples/)

Management API (/docs/alerting/0.20/management_api/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CAUTION: This page documents an old version of Alertmanager. Check out the latest stable version (/docs/alerting/latest/overview/).

NOTIFICATION TEMPLATE REFERENCE

- Data
- Alert
- KV
  - KV methods
- Strings

Prometheus creates and sends alerts to the Alertmanager which then sends notifications out to different receivers based on their labels. A receiver can be one of many integrations including: Slack, PagerDuty, email, or a custom integration via the generic webhook interface.

The notifications sent to receivers are constructed via templates. The Alertmanager comes with default templates but they can also be customized. To avoid confusion it's important to note that the Alertmanager templates differ from templating in Prometheus (/docs/visualization/template_reference/), however Prometheus templating also includes the templating in alert rule labels/annotations.

The Alertmanager's notification templates are based on the Go templating (https://golang.org/pkg/text/template) system. Note that some fields are evaluated as text, and others as HTML which will affect escaping.

DATA STRUCTURES

Data

`Data` is the structure passed to notification templates and webhook pushes.

Name	**Type**	**Notes**
Receiver	string	Defines the receiver's name that the notification will be sent to (slack, email etc.).
Status	string	Defined as firing if at least one alert is firing, otherwise resolved.
Alerts	Alert	List of all alert objects in this group (see below).
GroupLabels	KV	The labels these alerts were grouped by.
CommonLabels	KV	The labels common to all of the alerts.
CommonAnnotations	KV	Set of common annotations to all of the alerts. Used for longer additional strings of information about the alert.
ExternalURL	string	Backlink to the Alertmanager that sent the notification.

The `Alerts` type exposes functions for filtering alerts: - `Alerts.Firing` returns a list of currently firing alert objects in this group - `Alerts.Resolved` returns a list of resolved alert objects in this group

Alert

`Alert` holds one alert for notification templates.

Name	**Type**	**Notes**
Status	string	Defines whether or not the alert is resolved or currently firing.
Labels	KV	A set of labels to be attached to the alert.
Annotations	KV	A set of annotations for the alert.
StartsAt	time.Time	The time the alert started firing. If omitted, the current time is assigned by the Alertmanager.
EndsAt	time.Time	Only set if the end time of an alert is known. Otherwise set to a configurable timeout period from the time since the last alert was received.
GeneratorURL	string	A backlink which identifies the causing entity of this alert.

KV

`KV` is a set of key/value string pairs used to represent labels and annotations.

```
type KV map[string]string
```

Annotation example containing two annotations:

```
{
  summary: "alert summary",
  description: "alert description",
}
```

In addition to direct access of data (labels and annotations) stored as KV, there are also methods for sorting, removing, and viewing the LabelSets:

KV methods

Name	Arguments	Returns	Notes
SortedPairs	-	Pairs (list of key/value string pairs.)	Returns a sorted list of key/value pairs.
Remove	[]string	KV	Returns a copy of the key/value map without the given keys.
Names	-	[]string	Returns the names of the label names in the LabelSet.
Values	-	[]string	Returns a list of the values in the LabelSet.

FUNCTIONS

Note the default functions (https://golang.org/pkg/text/template/#hdr-Functions) also provided by Go templating.

Strings

Name	Arguments	Returns	Notes
title	string	strings.Title (https://golang.org/pkg/strings/#Title), capitalises first character of each word.	
toUpper	string	strings.ToUpper (https://golang.org/pkg/strings/#ToUpper), converts all characters to upper case.	
toLower	string	strings.ToLower (https://golang.org/pkg/strings/#ToLower), converts all characters to lower case.	
match	pattern, string	Regexp.MatchString (https://golang.org/pkg/regexp/#MatchString). Match a string using Regexp.	
reReplaceAll	pattern, replacement, text	Regexp.ReplaceAllString (https://golang.org/pkg/regexp/#Regexp.ReplaceAllString) Regexp substitution, unanchored.	

Name	**Arguments**	**Returns**	**Notes**
join	sep string, s []string	strings.Join (https://golang.org/pkg/strings/#Join), concatenates the elements of s to create a single string. The separator string sep is placed between elements in the resulting string. (note: argument order inverted for easier pipelining in templates.)	
safeHtml	text string	html/template.HTML (https://golang.org/pkg/html/template/#HTML), Marks string as HTML not requiring auto-escaping.	
stringSlice	...string	Returns the passed strings as a slice of strings.	

This documentation is open-source (https://github.com/prometheus/docs#contributing-changes). Please help improve it by filing issues or pull requests.