SPECIFICATIONS

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

SPECIFICATIONS

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.

SPECIFICATIONS

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: latest (0.27)

Alerting overview (/docs/alerting/latest/overview/)

Alertmanager (/docs/alerting/latest/alertmanager/)

Configuration (/docs/alerting/latest/configuration/)

Clients (/docs/alerting/latest/clients/)

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

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

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

HTTPS and authentication (/docs/alerting/latest/https/)

BEST PRACTICES

GUIDES

TUTORIALS

SPECIFICATIONS

CONFIGURATION

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.

- `<wechat_config>`
- `<webex_config>`

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 an HTTP POST request to the `/-/reload` endpoint.

Configuration file introduction

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]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?|0)`, e.g. `1d`, `1h30m`, `5m`, `10s`
- `<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
- `<int>`: an integer value

- `<regex>` : any valid RE2 regular expression (https://github.com/google/re2/wiki/Syntax) (The regex is anchored on both ends. To un-anchor the regex, use `.*<regex>.*` .)

The other placeholders are specified separately.

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

File layout and global settings

The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections. The other top-level sections are documented below on this page.

```
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 ST
  # 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 auther
  [ smtp_auth_username: <string> ]
  # SMTP Auth using LOGIN and PLAIN.
  [ smtp_auth_password: <secret> ]
  # SMTP Auth using LOGIN and PLAIN.
  [ smtp_auth_password_file: <string> ]
  # 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> ]
  [ slack_api_url_file: <filepath> ]
  [ victorops_api_key: <secret> ]
  [ victorops_api_key_file: <filepath> ]
  [ victorops_api_url: <string> | default = "https://alert.victorops.com/integratior
  [ pagerduty_url: <string> | default = "https://events.pagerduty.com/v2/enqueue" ]
  [ opsgenie_api_key: <secret> ]
  [ opsgenie_api_key_file: <filepath> ]
  [ opsgenie_api_url: <string> | default = "https://api.opsgenie.com/" ]
  [ wechat_api_url: <string> | default = "https://qyapi.weixin.qq.com/cgi-bin/" ]
  [ wechat_api_secret: <secret> ]
  [ wechat_api_corp_id: <string> ]
  [ telegram_api_url: <string> | default = "https://api.telegram.org" ]
  [ webex_api_url: <string> | default = "https://webexapis.com/v1/messages" ]
  # 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
  # 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> ... ]

# DEPRECATED: use time_intervals below.
# A list of mute time intervals for muting routes.
mute_time_intervals:
  [ - <mute_time_interval> ... ]

# A list of time intervals for muting/activating routes.
time_intervals:
  [ - <time_interval> ... ]
```

Route-related settings

Routing-related settings allow configuring how alerts are routed, aggregated, throttled, and muted based on time.

`<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.

See Alertmanager concepts (/docs/alerting/alertmanager/#grouping) for more information on grouping.

```
[ 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
# 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 ]

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

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

# A list of matchers that an alert has to fulfill to match the node.
matchers:
  [ - <matcher> ... ]

# 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.)
# If omitted, child routes inherit the group_wait of the parent route.
[ 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.) If omitted, child routes
# inherit the group_interval of the parent route.
[ 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). If omitted,
# child routes inherit the repeat_interval of the parent route.
# Note that this parameter is implicitly bound by Alertmanager's
# `--data.retention` configuration flag. Notifications will be resent after either
# repeat_interval or the data retention period have passed, whichever
# occurs first. `repeat_interval` should be a multiple of `group_interval`.
[ repeat_interval: <duration> | default = 4h ]

# Times when the route should be muted. These must match the name of a
# mute time interval defined in the mute_time_intervals section.
# Additionally, the root node cannot have any mute times.
# When a route is muted it will not send any notifications, but
# otherwise acts normally (including ending the route-matching process
# if the `continue` option is not set.)
mute_time_intervals:
  [ - <string> ...]

# Times when the route should be active. These must match the name of a
# time interval defined in the time_intervals section. An empty value
# means that the route is always active.
# Additionally, the root node cannot have any active times.
# The route will send notifications only when active, but otherwise
# acts normally (including ending the route-matching process
# if the `continue` option is not set).
active_time_intervals:
  [ - <string> ...]

# 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
    matchers:
    - 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]
    matchers:
    - team="frontend"

  # All alerts with the service=inhouse-service label match this sub-route.
  # the route will be muted during offhours and holidays time intervals.
  # even if it matches, it will continue to the next sub-route
  - receiver: 'dev-pager'
    matchers:
      - service="inhouse-service"
    mute_time_intervals:
      - offhours
      - holidays
    continue: true

    # All alerts with the service=inhouse-service label match this sub-route
    # the route will be active only during offhours and holidays time intervals.
  - receiver: 'on-call-pager'
    matchers:
      - service="inhouse-service"
    active_time_intervals:
      - offhours
      - holidays
```

`<time_interval>`

A `time_interval` specifies a named interval of time that may be referenced in the routing tree to mute/activate particular routes for particular times of the day.

```
name: <string>
time_intervals:
  [ - <time_interval_spec> ... ]
```

`<time_interval_spec>`

A `time_interval_spec` contains the actual definition for an interval of time. The syntax supports the following fields:

```
- times:
  [ - <time_range> ...]
  weekdays:
  [ - <weekday_range> ...]
  days_of_month:
  [ - <days_of_month_range> ...]
  months:
  [ - <month_range> ...]
  years:
  [ - <year_range> ...]
  location: <string>
```

All fields are lists. Within each non-empty list, at least one element must be satisfied to match the field. If a field is left unspecified, any value will match the field. For an instant of time to match a complete time interval, all fields must match. Some fields support ranges and negative indices, and are detailed below. If a time zone is not specified, then the times are taken to be in UTC.

`time_range`: Ranges inclusive of the starting time and exclusive of the end time to make it easy to represent times that start/end on hour boundaries. For example, `start_time: '17:00'` and `end_time: '24:00'` will begin at 17:00 and finish immediately before 24:00. They are specified like so:

```
times:
- start_time: HH:MM
  end_time: HH:MM
```

`weekday_range`: A list of days of the week, where the week begins on Sunday and ends on Saturday. Days should be specified by name (e.g. 'Sunday'). For convenience, ranges are also accepted of the form `<start_day>:<end_day>` and are inclusive on both ends. For example: `['monday:wednesday','saturday', 'sunday']`

`days_of_month_range`: A list of numerical days in the month. Days begin at 1. Negative values are also accepted which begin at the end of the month, e.g. -1 during January would represent January 31. For example: `['1:5', '-3:-1']`. Extending past the start or end of the month will cause it to be clamped. E.g. specifying `['1:31']` during February will clamp the actual end date to 28 or 29 depending on leap years. Inclusive on both ends.

`month_range`: A list of calendar months identified by a case-insensitive name (e.g. 'January') or by number, where January = 1. Ranges are also accepted. For example, `['1:3', 'may:august', 'december']`. Inclusive on both ends.

`year_range`: A numerical list of years. Ranges are accepted. For example, `['2020:2022', '2030']`. Inclusive on both ends.

`location`: A string that matches a location in the IANA time zone database. For example, `'Australia/Sydney'`. The location provides the time zone for the time interval. For example, a time interval with a location of `'Australia/Sydney'` that contained something like:

```
times:
- start_time: 09:00
  end_time: 17:00
weekdays: ['monday:friday']
```

would include any time that fell between the hours of 9:00AM and 5:00PM, between Monday and Friday, using the local time in Sydney, Australia.

You may also use `'Local'` as a location to use the local time of the machine where Alertmanager is running, or `'UTC'` for UTC time. If no timezone is provided, the time interval is taken to be in UTC time.**Note:** On Windows, only `Local` or `UTC` are supported unless you provide a custom time zone database using the `ZONEINFO` environment variable.

Inhibition-related settings

Inhibition allows muting a set of alerts based on the presence of another set of alerts. This allows establishing dependencies between systems or services such that only the most relevant of a set of interconnected alerts are sent out during an outage.

See Alertmanager concepts (/docs/alerting/alertmanager/#inhibition) for more information on inhibition.

`<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.

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

# A list of matchers that have to be fulfilled by the target
# alerts to be muted.
target_matchers:
  [ - <matcher> ... ]

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

# A list of matchers for which one or more alerts have
# to exist for the inhibition to take effect.
source_matchers:
  [ - <matcher> ... ]

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

Label matchers

Label matchers match alerts to routes, silences, and inhibition rules.

Important: Prometheus is adding support for UTF-8 in labels and metrics. In order to also support UTF-8 in the Alertmanager, Alertmanager versions 0.27 and later have a new parser for matchers that has a number of backwards incompatible changes. While most matchers will be forward-compatible, some will not.

Alertmanager is operating a transition period where it supports both UTF-8 and classic matchers, and has provided a number of tools to help you prepare for the transition.

If this is a new Alertmanager installation, we recommend enabling UTF-8 strict mode before creating an Alertmanager configuration file. You can find instructions on how to enable UTF-8 strict mode here.

If this is an existing Alertmanager installation, we recommend running the Alertmanager in the default mode called fallback mode before enabling UTF-8 strict mode. In this mode, Alertmanager will log a warning if you need to make any changes to your configuration file before UTF-8 strict mode can be enabled. Alertmanager will make UTF-8 strict mode the default in the next two versions, so it's important to transition as soon as possible.

Irrespective of whether an Alertmanager installation is a new or existing installation, you can also use `amtool` to validate that an Alertmanager configuration file is compatible with UTF-8 strict mode before enabling it in Alertmanager server. You do not need a running Alertmanager server to do this. You can find instructions on how to validate an Alertmanager configuration file using `amtool` here.

Alertmanager server operational modes

During the transition period, Alertmanager supports three modes of operation. These are known as fallback mode, UTF-8 strict mode and classic mode. Fallback mode is the default mode.

Operators of Alertmanager servers should transition to UTF-8 strict mode before the end of the transition period. Alertmanager will make UTF-8 strict mode the default in the next two versions, so it's important to transition as soon as possible.

Fallback mode

Alertmanager runs in a special mode called fallback mode as its default mode. As operators, you should not experience any difference in how your routes, silences or inhibition rules work.

In fallback mode, configurations are first parsed as UTF-8 matchers, and if incompatible with the UTF-8 parser, are then parsed as classic matchers. If your Alertmanager configuration contains matchers that are incompatible with the UTF-8 parser, Alertmanager will parse them as classic matchers and log a warning. This warning also includes a suggestion on how to change the matchers from classic matchers to UTF-8 matchers. For example:

```
ts=2024-02-11T10:00:00Z caller=parse.go:176 level=warn
msg="Alertmanager is moving to a new parser for labels and
matchers, and this input is incompatible. Alertmanager has instead
parsed the input using the classic matchers parser as a fallback. To
make this input compatible with the UTF-8 matchers parser please
make sure all regular expressions and values are double-quoted. If
you are still seeing this message please open an issue." input="foo="
origin=config err="end of input: expected label value"
suggestion="foo=\"\""
```

Here the matcher `foo=` can be made into a valid UTF-8 matcher by double quoting the right hand side of the expression to give `foo=""`. These two matchers are equivalent, however with UTF-8 matchers the right hand side of the matcher is a required field.

In rare cases, a configuration can cause disagreement between the UTF-8 and classic parser. This happens when a matcher is valid in both parsers, but due to added support for UTF-8, results in different parsings depending on which parser is used. If your Alertmanager configuration has disagreement, Alertmanager will use the classic parser and log a warning. For example:

```
ts=2024-02-11T10:00:00Z caller=parse.go:183 level=warn
msg="Matchers input has disagreement"
input="qux=\"\xf0\x9f\x99\x82\"\n" origin=config
```

Any occurrences of disagreement should be looked at on a case by case basis as depending on the nature of the disagreement, the configuration might not need updating before enabling UTF-8 strict mode. For example `\xf0\x9f\x99\x82` is the byte sequence for the 🙂 emoji. If the intention is to match a literal 🙂 emoji then no change is required. However, if the intention is to match the literal `\xf0\x9f\x99\x82` then the matcher should be changed to `qux="\\xf0\\x9f\\x99\\x82"`.

UTF-8 strict mode

In UTF-8 strict mode, Alertmanager disables support for classic matchers:

```
alertmanager --config.file=config.yml --enable-feature="utf8-strict-mode"
```

This mode should be enabled for new Alertmanager installations, and existing Alertmanager installations once all warnings of incompatible matchers have been resolved. Alertmanager will not start in UTF-8 strict mode until all the warnings of incompatible matchers have been resolved:

```
ts=2024-02-11T10:00:00Z caller=coordinator.go:118 level=error component=configuration msg="Loading configuration file failed" file=config.yml err="end of input: expected label value"
```

UTF-8 strict mode will be the default mode of Alertmanager at the end of the transition period.

Classic mode

Classic mode is equivalent to Alertmanager versions 0.26.0 and older:

```
alertmanager --config.file=config.yml --enable-feature="classic-mode"
```

You can use this mode if you suspect there is an issue with fallback mode or UTF-8 strict mode. In such cases, please open an issue on GitHub with as much information as possible.

Verification

You can use `amtool` to validate that an Alertmanager configuration file is compatible with UTF-8 strict mode before enabling it in Alertmanager server. You do not need a running Alertmanager server to do this.

Just like Alertmanager server, `amtool` will log a warning if the configuration is incompatible or contains disagreement:

```
amtool check-config config.yml
Checking 'config.yml'
level=warn msg="Alertmanager is moving to a new parser for labels and matchers, and
level=warn msg="Matchers input has disagreement" input="qux=\"\\xf0\\x9f\\x99\\x82\"
  SUCCESS
Found:
 - global config
 - route
 - 2 inhibit rules
 - 2 receivers
 - 0 templates
```

You will know if a configuration is compatible with UTF-8 strict mode when no warnings are logged in `amtool`:

```
amtool check-config config.yml
Checking 'config.yml'  SUCCESS
Found:
 - global config
 - route
 - 2 inhibit rules
 - 2 receivers
 - 0 templates
```

You can also use `amtool` in UTF-8 strict mode as an additional level of verification. You will know that a configuration is invalid because the command will fail:

```
amtool check-config config.yml --enable-feature="utf8-strict-mode"
level=warn msg="UTF-8 mode enabled"
Checking 'config.yml'  FAILED: end of input: expected label value

amtool: error: failed to validate 1 file(s)
```

You will know that a configuration is valid because the command will succeed:

```
amtool check-config config.yml --enable-feature="utf8-strict-mode"
level=warn msg="UTF-8 mode enabled"
Checking 'config.yml'  SUCCESS
Found:
 - global config
 - route
 - 2 inhibit rules
 - 2 receivers
 - 0 templates
```

`<matcher>`

UTF-8 matchers

A UTF-8 matcher consists of three tokens:

- An unquoted literal or a double-quoted string for the label name.
- One of `=`, `!=`, `=~`, or `!~`. `=` means equals, `!=` means not equal, `=~` means matches the regular expression and `!~` means doesn't match the regular expression.
- An unquoted literal or a double-quoted string for the regular expression or label value.

Unquoted literals can contain all UTF-8 characters other than the reserved characters. These are whitespace, and all characters in `{ } ! = ~ , \ " ' ``. For example, `foo`, `[a-zA-Z]+`, and `Προμηθεύς` (Prometheus in Greek) are all examples of valid unquoted literals. However, `foo!` is not a valid literal as `!` is a reserved character.

Double-quoted strings can contain all UTF-8 characters. Unlike unquoted literals, there are no reserved characters. You can even use UTF-8 code points. For example, `"foo!"`, `"bar,baz"`, `"\"baz qux\""` and `"\xf0\x9f\x99\x82"` are valid double-quoted strings.

Classic matchers

A classic matcher is a string with a syntax inspired by PromQL and OpenMetrics. The syntax of a classic matcher consists of three tokens:

- A valid Prometheus label name.
- One of `=`, `!=`, `=~`, or `!~`. `=` means equals, `!=` means that the strings are not equal, `=~` is used for equality of regex expressions and `!~` is used for un-equality of regex expressions. They have the same meaning as known from PromQL selectors.
- A UTF-8 string, which may be enclosed in double quotes. Before or after each token, there may be any amount of whitespace.

The 3rd token may be the empty string. Within the 3rd token, OpenMetrics escaping rules apply: `\"` for a double-quote, `\n` for a line feed, `\\` for a literal backslash. Unescaped `"` must not occur inside the 3rd token (only as the 1st or last character). However, literal line feed characters are tolerated, as are single `\` characters not followed by `\`, `n`, or `"`. They act as a literal backslash in that case.

Composition of matchers

You can compose matchers to create complex match expressions. When composed, all matchers must match for the entire expression to match. For example, the expression `{alertname="Watchdog", severity=~"warning|critical"}` will match an alert with labels `alertname=Watchdog, severity=critical` but not an alert with labels `alertname=Watchdog, severity=none` as while the alertname is Watchdog the severity is neither warning nor critical.

You can compose matchers into expressions with a YAML list:

```
matchers:
  - alertname = Watchdog
  - severity =~ "warning|critical"
```

or as a PromQL inspired expression where each matcher is comma separated:

```
{alertname="Watchdog", severity=~"warning|critical"}
```

A single trailing comma is permitted:

```
{alertname="Watchdog", severity=~"warning|critical",}
```

The open `{` and close `}` brace are optional:

```
alertname="Watchdog", severity=~"warning|critical"
```

However, both must be either present or omitted. You cannot have incomplete open or close braces:

```
{alertname="Watchdog", severity=~"warning|critical"
```

```
alertname="Watchdog", severity=~"warning|critical"}
```

You cannot have duplicate open or close braces either:

```
{{alertname="Watchdog", severity=~"warning|critical",}}
```

Whitespace (spaces, tabs and newlines) is permitted outside double quotes and has no effect on the matchers themselves. For example:

```
{
   alertname = "Watchdog",
   severity =~ "warning|critical",
}
```

is equivalent to:

```
{alertname="Watchdog",severity=~"warning|critical"}
```

More examples

Here are some more examples:

1. Two equals matchers composed as a YAML list:

```
matchers:
  - foo = bar
  - dings != bums
```

2. Two matchers combined composed as a short-form YAML list:

```
matchers: [ foo = bar, dings != bums ]
```

As shown below, in the short-form, it's better to use double quotes to avoid problems with special characters like commas:

```
matchers: [ "foo = \"bar,baz\"", "dings != bums" ]
```

1. You can also put both matchers into one PromQL-like string. Single quotes work best here:

```
matchers: [ '{foo="bar", dings!="bums"}' ]
```

2. To avoid issues with escaping and quoting rules in YAML, you can also use a YAML block:

```
matchers:
  - |
      {quote=~"She said: \"Hi, all!( How're you…)?\""}
```

General receiver-related settings

These receiver settings allow configuring notification destinations (receivers) and HTTP client options for HTTP-based receivers.

`<receiver>`

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

Note: As part of lifting the past moratorium on new receivers it was agreed that, in addition to the existing requirements, new notification integrations will be required to have a committed maintainer with push access.

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

# Configurations for several notification integrations.
discord_configs:
  [ - <discord_config>, ... ]
email_configs:
  [ - <email_config>, ... ]
msteams_configs:
  [ - <msteams_config>, ... ]
opsgenie_configs:
  [ - <opsgenie_config>, ... ]
pagerduty_configs:
  [ - <pagerduty_config>, ... ]
pushover_configs:
  [ - <pushover_config>, ... ]
slack_configs:
  [ - <slack_config>, ... ]
sns_configs:
  [ - <sns_config>, ... ]
telegram_configs:
  [ - <telegram_config>, ... ]
victorops_configs:
  [ - <victorops_config>, ... ]
webex_configs:
  [ - <webex_config>, ... ]
webhook_configs:
  [ - <webhook_config>, ... ]
wechat_configs:
  [ - <wechat_config>, ... ]
```

`<http_config>`

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

```
# Note that `basic_auth` and `authorization` 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> ]

# Optional the `Authorization` header configuration.
authorization:
  # Sets the authentication type.
  [ type: <string> | default: Bearer ]
  # Sets the credentials. It is mutually exclusive with
  # `credentials_file`.
  [ credentials: <secret> ]
  # Sets the credentials with the credentials read from the configured file.
  # It is mutually exclusive with `credentials`.
  [ credentials_file: <filename> ]

# Optional OAuth 2.0 configuration.
# Cannot be used at the same time as basic_auth or authorization.
oauth2:
  [ <oauth2> ]

# Whether to enable HTTP2.
[ enable_http2: <bool> | default: true ]

# Optional proxy URL.
[ proxy_url: <string> ]
# Comma-separated string that can contain IPs, CIDR notation, domain names
# that should be excluded from proxying. IP and domain names can
# contain port numbers.
[ no_proxy: <string> ]
# Use proxy URL indicated by environment variables (HTTP_PROXY, http_proxy, HTTPS_PR
[ proxy_from_environment: <boolean> | default: false ]
# Specifies headers to send to proxies during CONNECT requests.
[ proxy_connect_header:
  [ <string>: [<secret>, ...] ] ]

# Configure whether HTTP requests follow HTTP 3xx redirects.
[ follow_redirects: <bool> | default = true ]

# Configures the TLS settings.
```

```
tls_config:
  [ <tls_config> ]
```

`<oauth2>`

OAuth 2.0 authentication using the client credentials grant type. Alertmanager fetches an access token from the specified endpoint with the given client access and secret keys.

```
client_id: <string>
[ client_secret: <secret> ]

# Read the client secret from a file.
# It is mutually exclusive with `client_secret`.
[ client_secret_file: <filename> ]

# Scopes for the token request.
scopes:
  [ - <string> ... ]

# The URL to fetch the token from.
token_url: <string>

# Optional parameters to append to the token URL.
endpoint_params:
  [ <string>: <string> ... ]

# Configures the token request's TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]
# Comma-separated string that can contain IPs, CIDR notation, domain names
# that should be excluded from proxying. IP and domain names can
# contain port numbers.
[ no_proxy: <string> ]
# Use proxy URL indicated by environment variables (HTTP_PROXY, https_proxy, HTTPs_P
[ proxy_from_environment: <boolean> | default: false ]
# Specifies headers to send to proxies during CONNECT requests.
[ proxy_connect_header:
  [ <string>: [<secret>, ...] ] ]
```

`<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]

# Minimum acceptable TLS version. Accepted values: TLS10 (TLS 1.0), TLS11 (TLS
# 1.1), TLS12 (TLS 1.2), TLS13 (TLS 1.3).
# If unset, Prometheus will use Go default minimum version, which is TLS 1.2.
# See MinVersion in https://pkg.go.dev/crypto/tls#Config.
[ min_version: <string> ]
# Maximum acceptable TLS version. Accepted values: TLS10 (TLS 1.0), TLS11 (TLS
# 1.1), TLS12 (TLS 1.2), TLS13 (TLS 1.3).
# If unset, Prometheus will use Go default maximum version, which is TLS 1.3.
# See MaxVersion in https://pkg.go.dev/crypto/tls#Config.
[ max_version: <string> ]
```

Receiver integration settings

These settings allow configuring specific receiver integrations.

`<discord_config>`

Discord notifications are sent via the Discord webhook API (https://discord.com/developers/docs/resources/webhook). See Discord's "Intro to Webhooks" article (https://support.discord.com/hc/en-us/articles/228383668-Intro-to-Webhooks) to learn how to configure a webhook integration for a channel.

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The Discord webhook URL.
# webhook_url and webhook_url_file are mutually exclusive.
webhook_url: <secret>
webhook_url_file: <filepath>

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

# Message body template.
[ message: <tmpl_string> | default = '{{ template "discord.default.message" . }}' ]

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

<email_config>

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

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

# The sender's 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_password and auth_password_file are mutually exclusive.
[ auth_username: <string> | default = global.smtp_auth_username ]
[ auth_password: <secret> | default = global.smtp_auth_password ]
[ auth_password_file: <string> | default = global.smtp_auth_password_file ]
[ 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>, ... } ]
```

`<msteams_config>`

Microsoft Teams notifications are sent via the Incoming Webhooks (https://learn.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/what-are-webhooks-and-connectors) API endpoint.

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The incoming webhook URL.
# webhook_url and webhook_url_file are mutually exclusive.
[ webhook_url: <secret> ]
[ webhook_url_file: <filepath> ]

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

# Message summary template.
[ summary: <tmpl_string> | default = '{{ template "msteams.default.summary" . }}' ]

# Message body template.
[ text: <tmpl_string> | default = '{{ template "msteams.default.text" . }}' ]

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

`<opsgenie_config>`

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

```
# Whether 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 filepath to API key to use when talking to the OpsGenie API. Conflicts with ap
[ api_key_file: <filepath> | default = global.opsgenie_api_key_file ]

# 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> | default = '{{ template "opsgenie.default.message" . }}' ]

# A description of the alert.
[ 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 alert.
# All common labels are included as details by default.
[ 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> ]

# Whether to update message and description of the alert in OpsGenie if it already e
# By default, the alert is never updated in OpsGenie, the new message only appears i
[ update_alerts: <boolean> | default = false ]

# Optional field that can be used to specify which domain alert is related to.
```

```
[ entity: <tmpl_string> ]

# Comma separated list of actions that will be available for the alert.
[ actions: <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", "teams", "user", "escalation" or "schedule".
type: <tmpl_string>
```

<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 to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The routing and service keys are mutually exclusive.
# The PagerDuty integration key (when using PagerDuty integration type `Events API v
# It is mutually exclusive with `routing_key_file`.
routing_key: <tmpl_secret>
# Read the Pager Duty routing key from a file.
# It is mutually exclusive with `routing_key`.
routing_key_file: <filepath>
# The PagerDuty integration key (when using PagerDuty integration type `Prometheus`)
# It is mutually exclusive with `service_key_file`.
service_key: <tmpl_secret>
# Read the Pager Duty service key from a file.
# It is mutually exclusive with `service_key`.
service_key_file: <filepath>

# 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.descriptior

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

# Unique location of the affected system.
[ source: <tmpl_string> | default = client ]

# 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 part or component of the affected system that is broken.
[ component: <tmpl_string> ]

# A cluster or grouping of sources.
[ group: <tmpl_string> ]

# The class/type of the event.
[ class: <tmpl_string> ]

# 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://developer.pagerduty.com/docs/events-api-v2/trigger-events/#the-images-property).

```
href: <tmpl_string>
src: <tmpl_string>
alt: <tmpl_string>
```

`<link_config>`

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

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

`<pushover_config>`

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

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The recipient user's key.
# user_key and user_key_file are mutually exclusive.
user_key: <secret>
user_key_file: <filepath>

# Your registered application's API token, see https://pushover.net/apps
# You can also register a token by cloning this Prometheus app:
# https://pushover.net/apps/clone/prometheus
# token and token_file are mutually exclusive.
token: <secret>
token_file: <filepath>

# 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" . }}' ]

# Optional device to send notification to, see https://pushover.net/api#device
[ device: <string> ]

# Optional sound to use for notification, see https://pushover.net/api#sound
[ sound: <string> ]

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

# 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 ]

# Optional time to live (TTL) to use for notification, see https://pushover.net/api#
[ ttl: <duration> ]
```

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

`<slack_config>`

Slack notifications can be sent via Incoming webhooks (https://api.slack.com/messaging/webhooks) or Bot tokens (https://api.slack.com/authentication/token-types).

If using an incoming webhook then `api_url` must be set to the URL of the incoming webhook, or written to the file referenced in `api_url_file` .

If using Bot tokens then `api_url` must be set to `https://slack.com/api/chat.postMessage` (https://api.slack.com/methods/chat.postMessage), the bot token must be set as the authorization credentials in `http_config` , and `channel` must contain either the name of the channel or Channel ID to send notifications to. If using the name of the channel the # is optional.

The notification contains an attachment (https://api.slack.com/messaging/composing/layouts#attachments).

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = false ]

# The Slack webhook URL. Either api_url or api_url_file should be set.
# Defaults to global settings if none are set here.
[ api_url: <secret> | default = global.slack_api_url ]
[ api_url_file: <filepath> | default = global.slack_api_url_file ]

# 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 }}gooc
[ 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/messaging/composing/layouts#attachments) and interactive messages (https://api.slack.com/legacy/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/legacy/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/messaging/composing/layouts#attachments).

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

<sns_config>

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The SNS API URL i.e. https://sns.us-east-2.amazonaws.com.
#  If not specified, the SNS API URL from the SNS SDK will be used.
[ api_url: <tmpl_string> ]

# Configures AWS's Signature Verification 4 signing process to sign requests.
sigv4:
  [ <sigv4_config> ]

# SNS topic ARN, i.e. arn:aws:sns:us-east-2:698519295917:My-Topic
# If you don't specify this value, you must specify a value for the phone_number or
# If you are using a FIFO SNS topic you should set a message group interval longer t
# to prevent messages with the same group key being deduplicated by the SNS default
[ topic_arn: <tmpl_string> ]

# Subject line when the message is delivered to email endpoints.
[ subject: <tmpl_string> | default = '{{ template "sns.default.subject" .}}' ]

# Phone number if message is delivered via SMS in E.164 format.
# If you don't specify this value, you must specify a value for the topic_arn or tar
[ phone_number: <tmpl_string> ]

# The  mobile platform endpoint ARN if message is delivered via mobile notifications
# If you don't specify this value, you must specify a value for the topic_arn or pho
[ target_arn: <tmpl_string> ]

# The message content of the SNS notification.
[ message: <tmpl_string> | default = '{{ template "sns.default.message" .}}' ]

# SNS message attributes.
attributes:
  [ <string>: <string> ... ]

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

<sigv4_config>

```
# The AWS region. If blank, the region from the default credentials chain is used.
[ region: <string> ]

# The AWS API keys. Both access_key and secret_key must be supplied or both must be
# If blank the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`
[ access_key: <string> ]
[ secret_key: <secret> ]

# Named AWS profile used to authenticate.
[ profile: <string> ]

# AWS Role ARN, an alternative to using AWS API keys.
[ role_arn: <string> ]
```

`<telegram_config>`

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The Telegram API URL i.e. https://api.telegram.org.
# If not specified, default API URL will be used.
[ api_url: <string> | default = global.telegram_api_url ]

# Telegram bot token. It is mutually exclusive with `bot_token_file`.
[ bot_token: <secret> ]

# Read the Telegram bot token from a file. It is mutually exclusive with `bot_token`
[ bot_token_file: <filepath> ]

# ID of the chat where to send the messages.
[ chat_id: <int> ]

# Message template.
[ message: <tmpl_string> default = '{{ template "telegram.default.message" .}}' ]

# Disable telegram notifications
[ disable_notifications: <boolean> | default = false ]

# Parse mode for telegram message, supported values are MarkdownV2, Markdown, HTML a
[ parse_mode: <string> | default = "HTML" ]

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

`<victorops_config>`

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

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The API key to use when talking to the VictorOps API.
# It is mutually exclusive with `api_key_file`.
[ api_key: <secret> | default = global.victorops_api_key ]

# Reads the API key to use when talking to the VictorOps API from a file.
# It is mutually exclusive with `api_key`.
[ api_key_file: <filepath> | default = global.victorops_api_key_file ]

# 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.ent

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

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

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

`<webhook_config>`

The webhook receiver allows configuring a generic receiver.

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The endpoint to send HTTP POST requests to.
# url and url_file are mutually exclusive.
url: <secret>
url_file: <filepath>

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

# The maximum number of alerts to include in a single webhook message. Alerts
# above this threshold are truncated. When leaving this at its default value of
# 0, all alerts are included.
[ max_alerts: <int> | default = 0 ]
```

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
  "truncatedAlerts": <int>,          // how many alerts have been truncated due to "
  "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
      "fingerprint": <string>        // fingerprint to identify 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 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" . }}' ]
# Type of the message type, supported values are `text` and `markdown`.
[ message_type: <string> | default = 'text' ]
[ 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" . }}' ]
```

<webex_config>

```
# Whether to notify about resolved alerts.
[ send_resolved: <boolean> | default = true ]

# The Webex Teams API URL i.e. https://webexapis.com/v1/messages
# If not specified, default API URL will be used.
[ api_url: <string> | default = global.webex_api_url ]

# ID of the Webex Teams room where to send the messages.
room_id: <string>

# Message template.
[ message: <tmpl_string> default = '{{ template "webex.default.message" .}}' ]

# The HTTP client's configuration. You must use this configuration to supply the bot
[ http_config: <http_config> | default = global.http_config ]
```

SPECIFICATIONS

HTTPS AND AUTHENTICATION

Alertmanager supports basic authentication and TLS. This is **experimental** and might change in the future.

- HTTP Traffic
- Gossip Traffic

Currently TLS is supported for the HTTP traffic and gossip traffic.

HTTP Traffic

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

The file is written in 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.

The file is read upon every http request, such as any change in the configuration and the certificates is picked up immediately.

Generic placeholders are defined as follows:

- `<boolean>` : a boolean that can take the values `true` or `false`
- `<filename>` : a valid path in the current working directory
- `<secret>` : a regular string that is a secret, such as a password
- `<string>` : a regular string

```
tls_server_config:
  # Certificate and key files for server to use to authenticate to client.
  cert_file: <filename>
  key_file: <filename>

  # Server policy for client authentication. Maps to ClientAuth Policies.
  # For more detail on clientAuth options:
  # https://golang.org/pkg/crypto/tls/#ClientAuthType
  #
  # NOTE: If you want to enable client authentication, you need to use
  # RequireAndVerifyClientCert. Other values are insecure.
  [ client_auth_type: <string> | default = "NoClientCert" ]

  # CA certificate for client certificate authentication to the server.
  [ client_ca_file: <filename> ]

  # Minimum TLS version that is acceptable.
  [ min_version: <string> | default = "TLS12" ]

  # Maximum TLS version that is acceptable.
  [ max_version: <string> | default = "TLS13" ]

  # List of supported cipher suites for TLS versions up to TLS 1.2. If empty,
  # Go default cipher suites are used. Available cipher suites are documented
  # in the go documentation:
  # https://golang.org/pkg/crypto/tls/#pkg-constants
  #
  # Note that only the cipher returned by the following function are supported:
  # https://pkg.go.dev/crypto/tls#CipherSuites
  [ cipher_suites:
    [ - <string> ] ]

  # prefer_server_cipher_suites controls whether the server selects the
  # client's most preferred ciphersuite, or the server's most preferred
  # ciphersuite. If true then the server's preference, as expressed in
  # the order of elements in cipher_suites, is used.
  [ prefer_server_cipher_suites: <bool> | default = true ]

  # Elliptic curves that will be used in an ECDHE handshake, in preference
  # order. Available curves are documented in the go documentation:
  # https://golang.org/pkg/crypto/tls/#CurveID
  [ curve_preferences:
    [ - <string> ] ]
```

```
http_server_config:
  # Enable HTTP/2 support. Note that HTTP/2 is only supported with TLS.
  # This can not be changed on the fly.
  [ http2: <boolean> | default = true ]
  # List of headers that can be added to HTTP responses.
  [ headers:
    # Set the Content-Security-Policy header to HTTP responses.
    # Unset if blank.
    [ Content-Security-Policy: <string> ]
    # Set the X-Frame-Options header to HTTP responses.
    # Unset if blank. Accepted values are deny and sameorigin.
    # https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
    [ X-Frame-Options: <string> ]
    # Set the X-Content-Type-Options header to HTTP responses.
    # Unset if blank. Accepted value is nosniff.
    # https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-O
    [ X-Content-Type-Options: <string> ]
    # Set the X-XSS-Protection header to all responses.
    # Unset if blank.
    # https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
    [ X-XSS-Protection: <string> ]
    # Set the Strict-Transport-Security header to HTTP responses.
    # Unset if blank.
    # Please make sure that you use this with care as this header might force
    # browsers to load Prometheus and the other applications hosted on the same
    # domain and subdomains over HTTPS.
    # https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-
    [ Strict-Transport-Security: <string> ] ]

# Usernames and hashed passwords that have full access to the web
# server via basic authentication. If empty, no basic authentication is
# required. Passwords are hashed with bcrypt.
basic_auth_users:
  [ <string>: <secret> ... ]
```

Gossip Traffic

To specify whether to use mutual TLS for gossip, use the `--cluster.tls-config` flag.

The server and client sides of the gossip are configurable.

```
tls_server_config:
  # Certificate and key files for server to use to authenticate to client.
  cert_file: <filename>
  key_file: <filename>

  # Server policy for client authentication. Maps to ClientAuth Policies.
  # For more detail on clientAuth options:
  # https://golang.org/pkg/crypto/tls/#ClientAuthType
  [ client_auth_type: <string> | default = "NoClientCert" ]

  # CA certificate for client certificate authentication to the server.
  [ client_ca_file: <filename> ]

  # Minimum TLS version that is acceptable.
  [ min_version: <string> | default = "TLS12" ]

  # Maximum TLS version that is acceptable.
  [ max_version: <string> | default = "TLS13" ]

  # List of supported cipher suites for TLS versions up to TLS 1.2. If empty,
  # Go default cipher suites are used. Available cipher suites are documented
  # in the go documentation:
  # https://golang.org/pkg/crypto/tls/#pkg-constants
  [ cipher_suites:
    [ - <string> ] ]

  # prefer_server_cipher_suites controls whether the server selects the
  # client's most preferred ciphersuite, or the server's most preferred
  # ciphersuite. If true then the server's preference, as expressed in
  # the order of elements in cipher_suites, is used.
  [ prefer_server_cipher_suites: <bool> | default = true ]

  # Elliptic curves that will be used in an ECDHE handshake, in preference
  # order. Available curves are documented in the go documentation:
  # https://golang.org/pkg/crypto/tls/#CurveID
  [ curve_preferences:
    [ - <string> ] ]

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

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

  # Server name 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]
```

SPECIFICATIONS

MANAGEMENT API

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

- Health check
- Readiness check
- Reload

Health check

```
GET /-/healthy
HEAD /-/healthy
```

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

Readiness check

```
GET /-/ready
HEAD /-/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.

SPECIFICATIONS

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

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:
  # Also possible to place this URL in a file.
  # Ex: `slack_api_url_file: '/etc/alertmanager/slack_url'`
  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 down
      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.text":

```
{{ 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/).

NOTIFICATION TEMPLATE REFERENCE

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.
Fingerprint	string	Fingerprint that can be used to identify the 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.	
trimSpace	string	strings.TrimSpace (https://pkg.go.dev/strings#TrimSpace), removes leading and trailing white spaces.	
match	pattern, string	Regexp.MatchString (https://golang.org/pkg/regexp/#MatchString). Match a string using Regexp.	

Name	Arguments	Returns	Notes
reReplaceAll	pattern, replacement, text	Regexp.ReplaceAllString (https://golang.org/pkg/regexp/#Regexp.ReplaceAllString) Regexp substitution, unanchored.	
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.	