karma

Alert dashboard for Prometheus Alertmanager

View on GitHub

karma

Alert dashboard for Prometheus Alertmanager.

Alertmanager `>=0.22.0` is required.

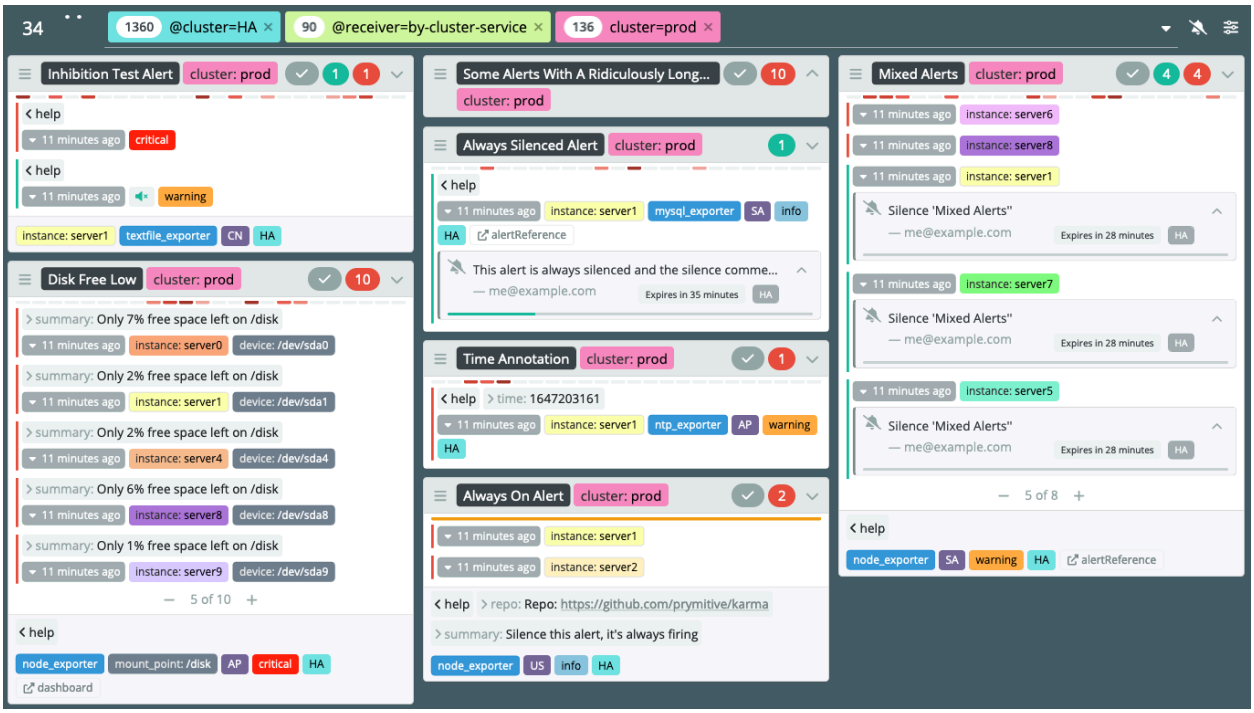
See GitHub Releases for release changelog.

Feature overview

Alertmanager UI is useful for browsing alerts and managing silences, but it's lacking as a dashboard tool - karma aims to fill this gap.

Alert aggregation and deduplication

Starting with the `0.7.0` release it can aggregate alerts from multiple Alertmanager instances, running either in HA mode or separate. Unique alerts are displayed by filtering duplicates. Each alert is tagged with the names of all Alertmanager instances it was found at and can be filtered based on those tags (`@alertmanager`). Note that `@alertmanager` tags will be visible only if karma is configured with multiple Alertmanager instances. If alertmanger is configured to use HA clusters then `@cluster` will be available as well, to set a custom name for each cluster see CONFIGURATION.md.

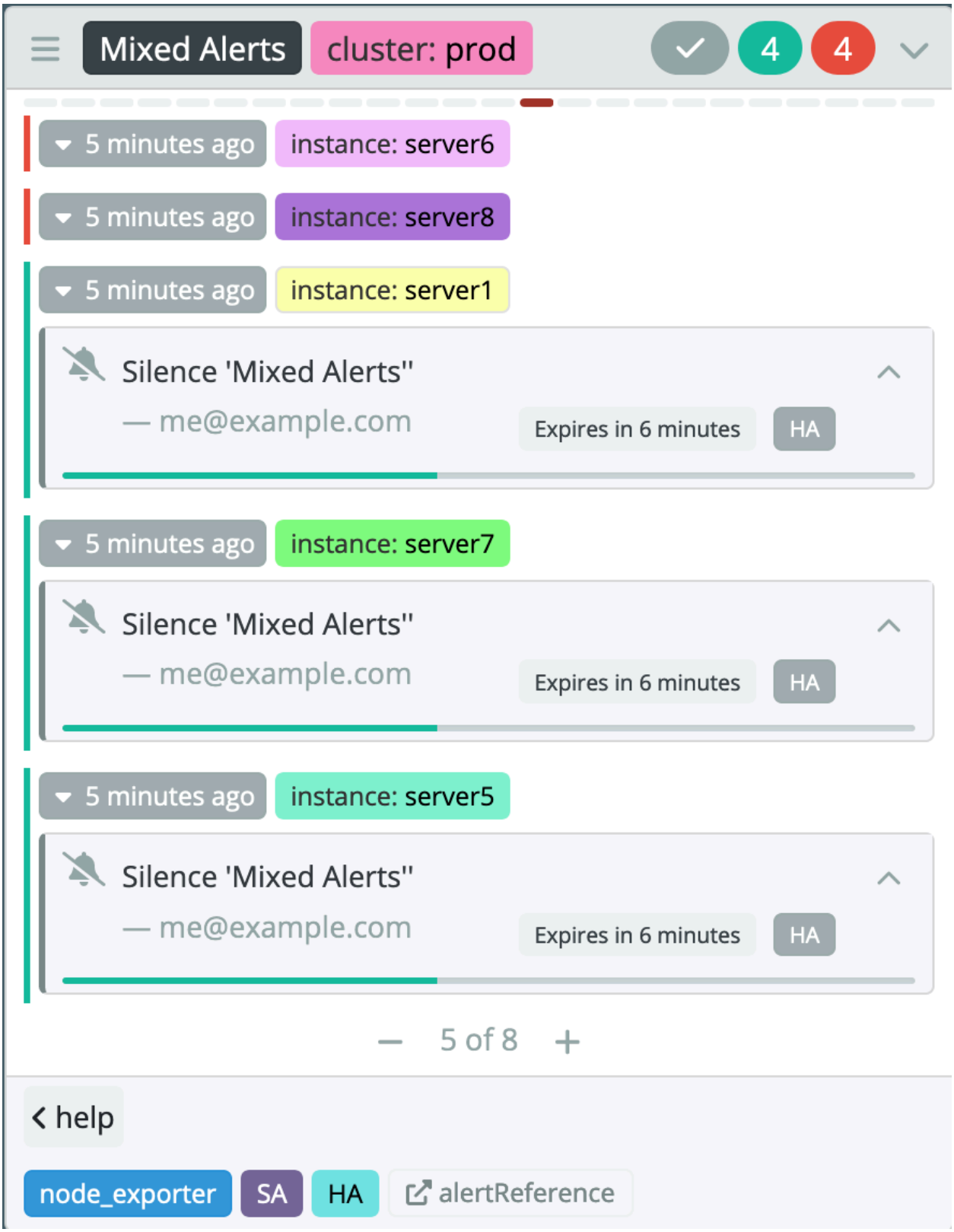


Alert visualization

Alert groups

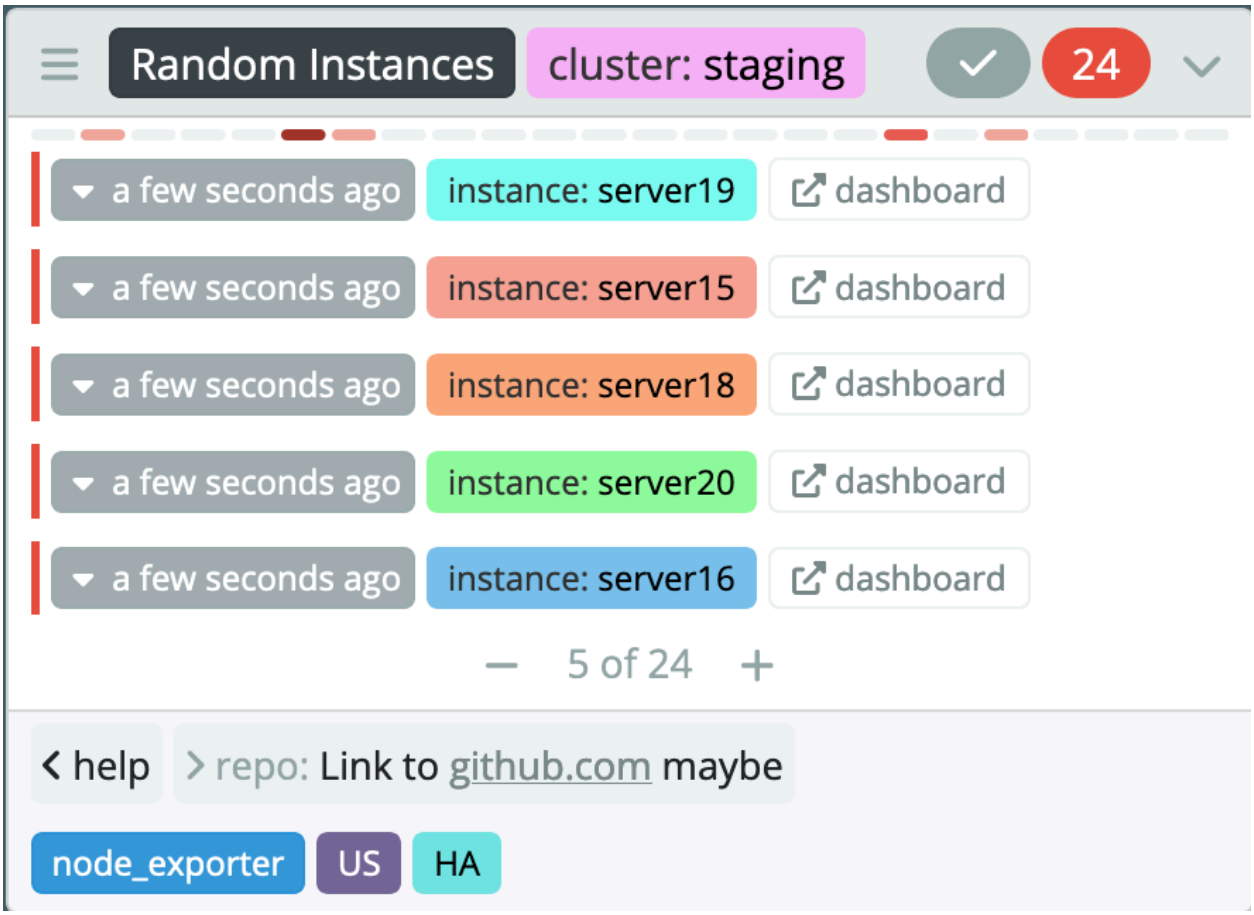
Alerts are displayed grouped preserving group_by configuration option in Alertmanager. Note that a unique alert group will be created for each receiver it uses in alertmanager as they can have different `group_by` settings. If a group contains multiple alerts only the first few alerts will be presented. Alerts are expanded or hidden using - / + buttons. The default number of alerts can be configured in the UI settings module. Each group can be collapsed to only show the title bar using top right toggle icon. Each individual alert will show unique labels and annotations. Labels and annotations that are shared between all alerts are moved to the footer.

Active alerts will show recently expired silences, to allow re-silincing if needed. This is controlled via `silences:expired` setting. `10m` value would show silences expired in the last 10 minutes but only for alerts that started firing more than 10 minutes ago.


Mixed Alerts
cluster: prod
4
4
5 minutes ago
instance: server6
5 minutes ago
instance: server8
5 minutes ago
instance: server1
Silence 'Mixed Alerts''
— me@example.com
Expires in 6 minutes
HA
5 minutes ago
instance: server7
Silence 'Mixed Alerts''
— me@example.com
Expires in 6 minutes
HA
5 minutes ago
instance: server5
Silence 'Mixed Alerts''
— me@example.com
Expires in 6 minutes
HA
5 of 8
help
node_exporter
SA
HA
alertReference

Alert history

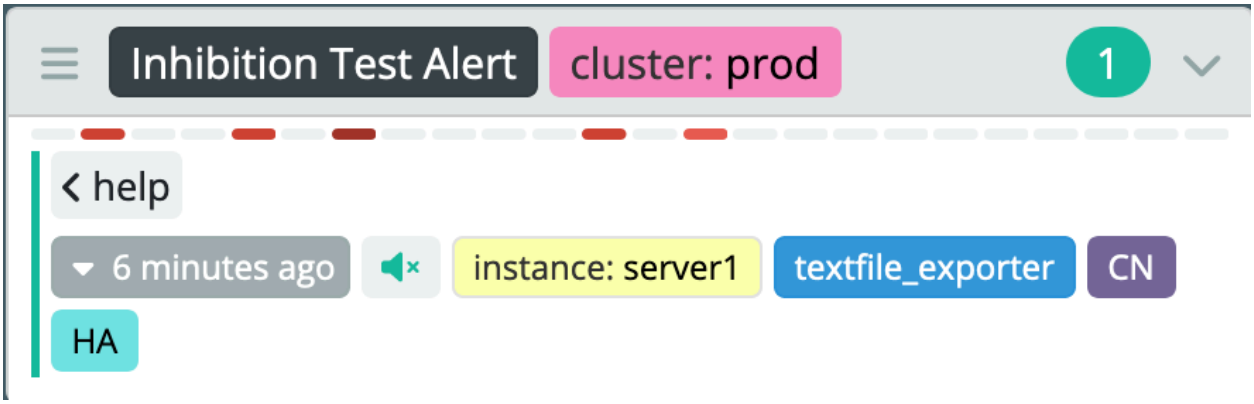
Alertmanager doesn't currently provide any long term storage of alert events or a way to query for historical alerts, but each Prometheus server sending alerts stores metrics related to triggered alerts. When `history:enabled` is `true` karma will use `source` fields from each alert to try querying alert related metrics on remote Prometheus servers. The result is the number of times given alert group triggered an alert per hour in the last 24h, displayed as 24 blocks. The darker the color the more alerts were triggered in that hour, as compared by all other hours.



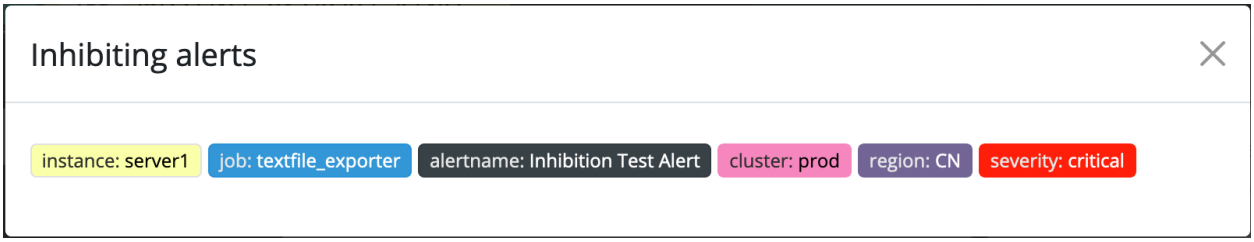
For this feature to work karma must be able to connect to all Prometheus servers sending alerts. Be sure to set `--web.external-url` Prometheus flag to a publicly reachable URL of each server.

Inhibited alerts

Inhibited alerts (suppressed by other alerts, see Alertmanager docs) will have a "muted" button.

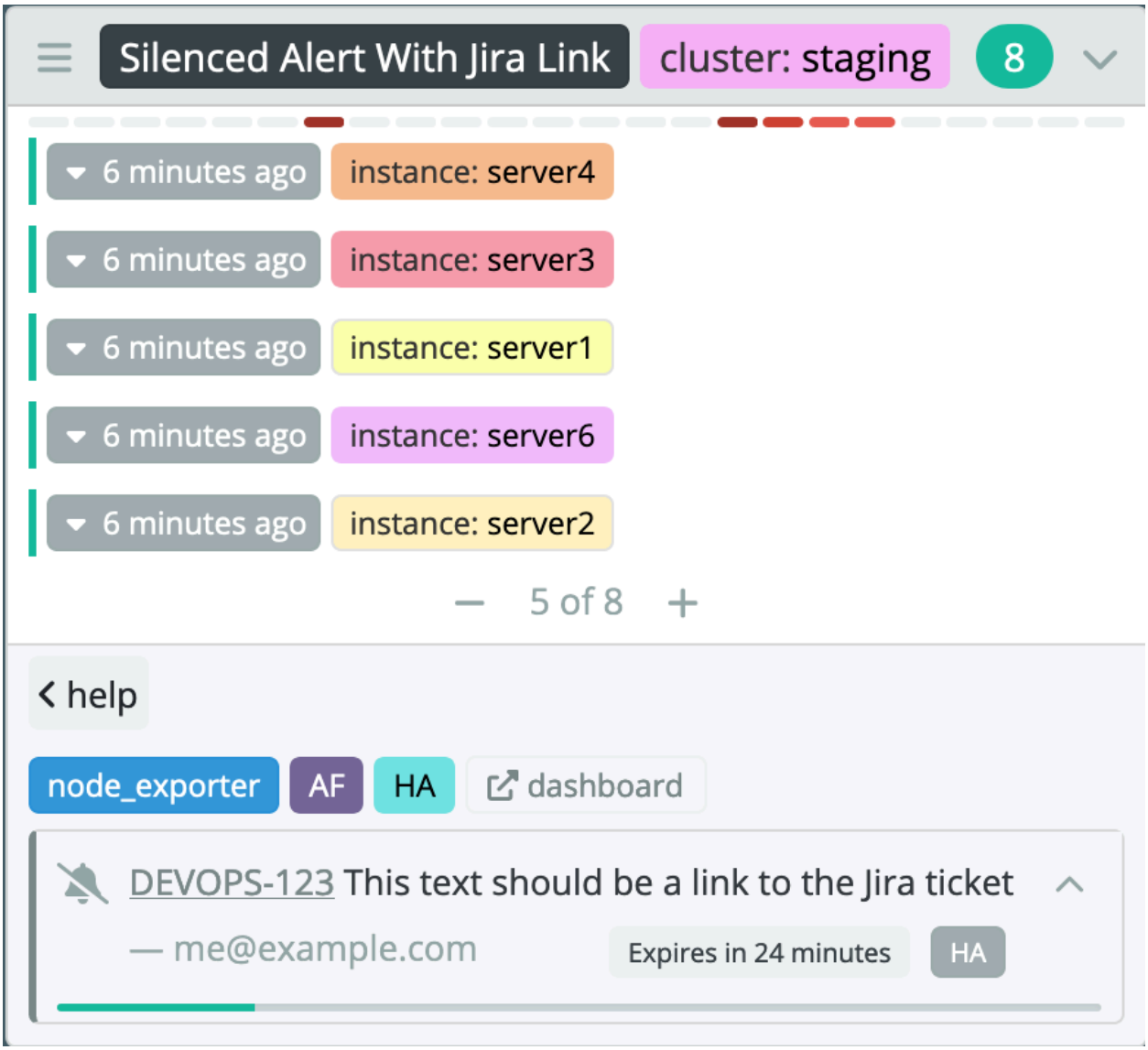


Clicking on that button will bring a modal with a list of inhibiting alerts.



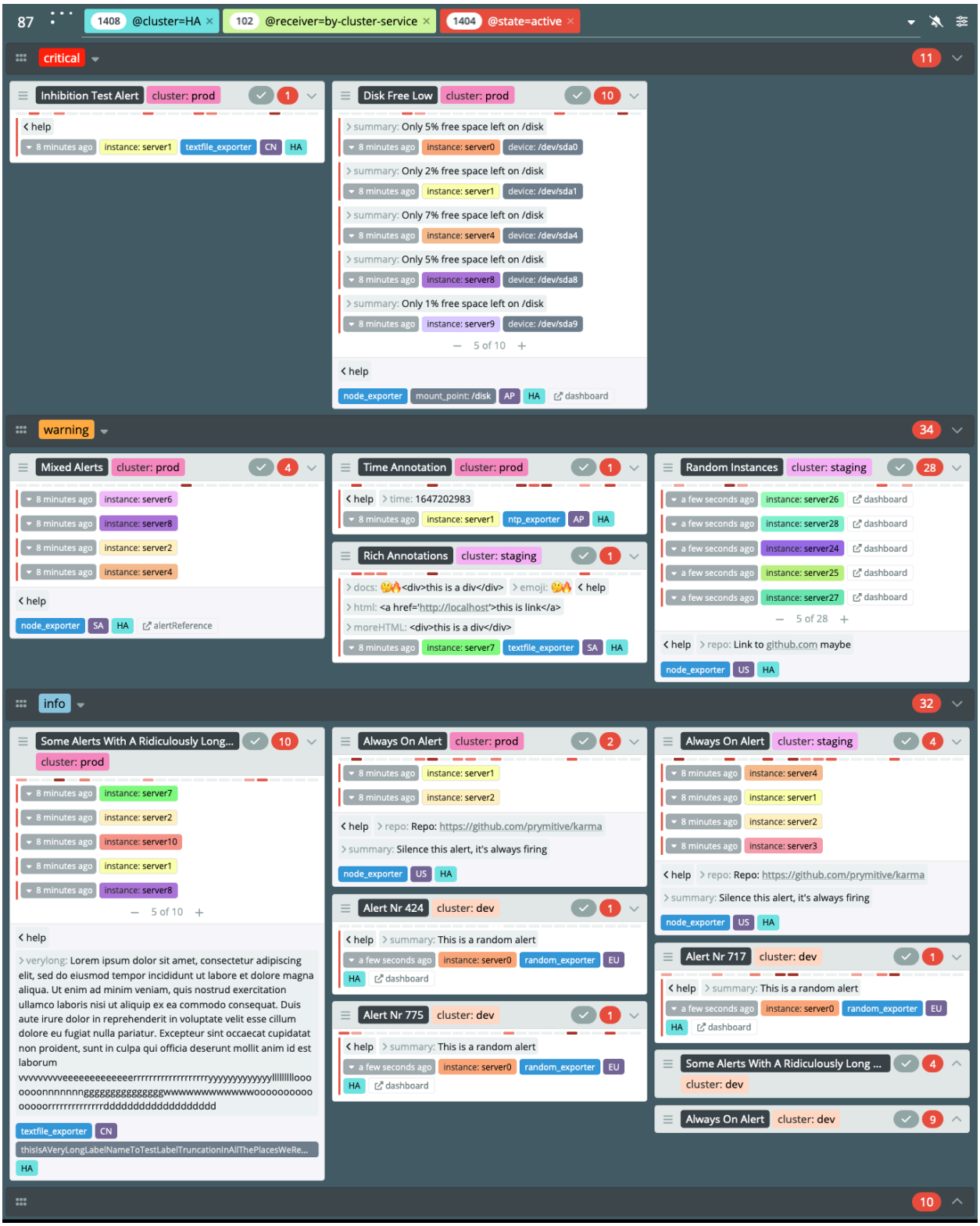
Silence deduplication

If all alerts in a group were suppressed by the same silence then, to save screen space, the silence will also be moved to the footer.



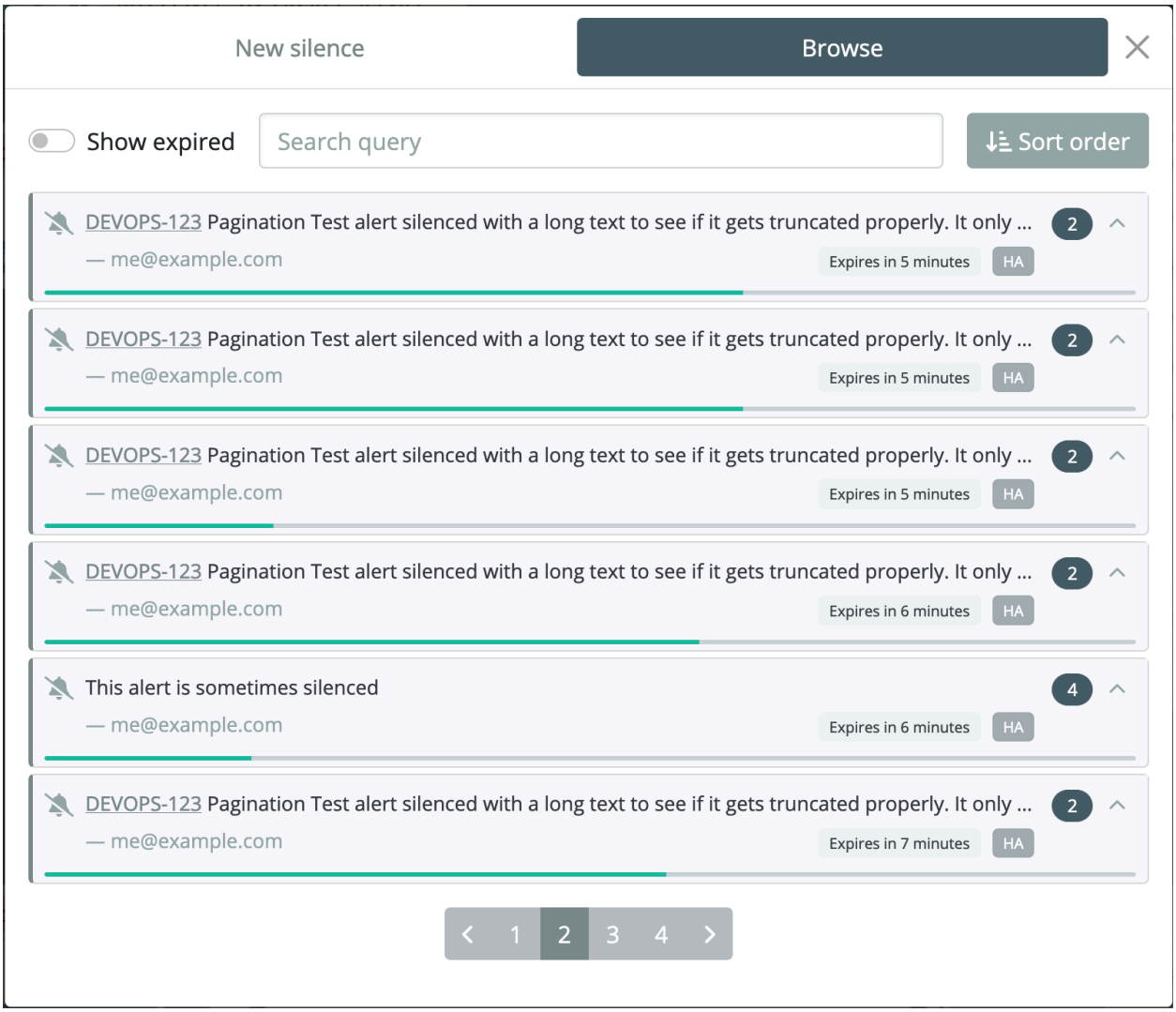
Label based multi-grid

To help separate alerts from different environments or with different level of severity multi-grid mode can be enabled, which adds another layer of visually grouping alert groups. To enable this mode go to the configuration modal and select a label name, all alerts will be grouped by that label, each label value will have a dedicated grid, including an extra grid for alerts without that label present.



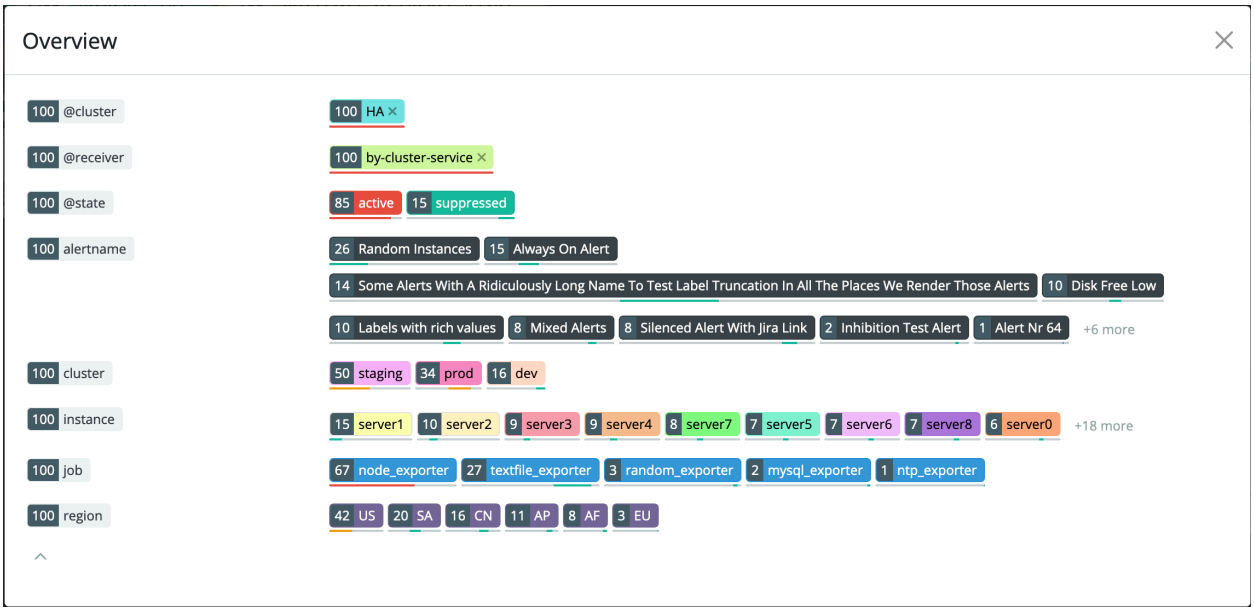
Silence management

Silence modal allows to create new silences and manage all silences already present in Alertmanager. Silence ACL rules can be used to control silence creation and editing, see ACLs docs for more details.



Alert overview

Clicking on the alert counter in the top left corner will open the overview modal, which allows to quickly get an overview of the top label values for all current alerts.



Alert acknowledgement

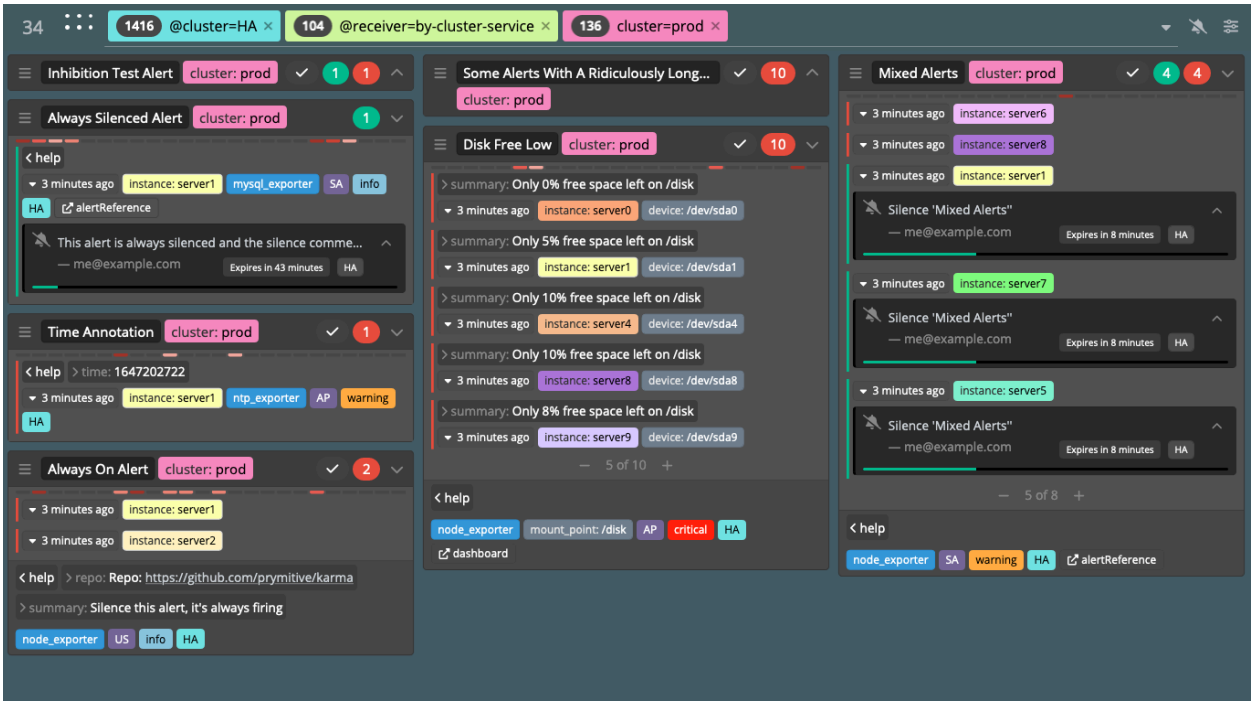
Starting with `v0.50` karma can create short lived silences to acknowledge alerts with a single button click. To create silences that will resolve itself only after all alerts are resolved you can use kthxbye. See configuration docs for details.

Dead Man's Switch support

Starting with `v0.78` karma can be configured to check for Dead Man's Switch style alerts (alert that is always firing). If no alert is found in given alertmanager karma will show an error in the UI. See `healthcheck:filters` option on configuration docs for details.

Dark mode

Starting with `v0.52` release karma includes both light and dark themes. By default it will follow browser preference using prefers-color-scheme media queries.



Demo

Online demo is running latest main branch. It might include features that are experimental and not yet ready to be included.

Release notes

Release notes can be found on GitHub Release Page.

To get notifications about new karma releases go to GitHub karma page, click `Watch` and select `Releases only`. This requires GitHub user account. To subscribe to email notifications without GitHub account you can subscribe to the RSS feed that GitHub provides. To get email notifications from those feeds use one of the free services providing *RSS to email* notifications, like Blogtrottr.

History

I created karma while working for Cloudflare, originally it was called unsee. This project is based on that code but the UI part was rewritten from scratch using

React. New UI required changes to the backend so the API is also incompatible. Given that the React rewrite resulted in roughly 50% of new code and to avoid confusion for user I've decided to rename it to karma, especially that the original project wasn't being maintained anymore.

Supported Alertmanager versions

Alertmanager's API isn't stable yet and can change between releases, see `VERSIONS` in internal/mock/Makefile for list of all Alertmanager releases that are tested and supported by karma. Due to API differences between those releases some features will work differently or be missing, it's recommended to use the latest supported Alertmanager version.

Security

karma doesn't in any way alter alerts in any Alertmanager instance it collects data from. This is true for both the backend and the web UI. The web UI allows to manage silences by sending requests to Alertmanager instances, this can be done directly (browser to Alertmanager API) or by proxying such requests via karma backend (browser to karma backend to Alertmanager API) if `proxy` mode is enabled in karma config.

If you wish to deploy karma as a read-only tool without giving users any ability to modify data in Alertmanager instance, then please ensure that:

- the karma process is able to connect to the Alertmanager API
- read-only users are able to connect to the karma web interface
- read-only users are NOT able to connect to the Alertmanager API
- `readonly` is set to `true` in alertmanager:servers config section for all alertmanager instances, this options will disable any UI elements that could trigger updates (like silence management)

To restrict some users from creating silences or enforce some matcher rules use silence ACL rules. This feature requires `proxy` to be enabled.

Metrics

karma process metrics are accessible under `/metrics` path by default. If you set the `--listen.prefix` option a path relative to it will be used.

Building and running

Building from source

To clone git repo and build the binary yourself run:

```
git clone https://github.com/prymitive/karma $GOPATH/src/github.com/prymitive
cd $GOPATH/src/github.com/prymitive/karma
```

To finally compile `karma` the binary run:

```
make
```

Note that building locally from sources requires Go, nodejs and yarn. See Docker build options below for instructions on building from withing docker container.

Running

`karma` can be configured using config file, command line flags or environment variables. Config file is the recommended method, it's also the only way to configure karma to use multiple Alertmanager servers for collecting alerts. To run karma with a single Alertmanager server set `ALERTMANAGER_URI` environment variable or pass `--alertmanger.uri` flag on the command line, with Alertmanager URI as argument, example:

```
ALERTMANAGER_URI=https://alertmanager.example.com karma
karma --alertmanager.uri https://alertmanager.example.com
```

There is a make target which will compile and run a demo karma docker image:

```
make run-demo
```

By default it will listen on port `8080` and will have mock alerts.

Docker

Running pre-build docker image

Official docker images are built and hosted on Github.

Images are built automatically for:

- release tags in git - `ghcr.io/prymitive/karma:vX.Y.Z`
- main branch commits - `ghcr.io/prymitive/karma:latest`

NOTE karma uses uber-go/automaxprocs to automatically adjust `GOMAXPROCS` to match Linux container CPU quota.

Examples

To start a release image run:

```
docker run -e ALERTMANAGER_URI=https://alertmanager.example.com ghcr.io/prymi
```

Latest release details can be found on GitHub.

To start docker image build from lastet main branch run:

```
docker run -e ALERTMANAGER_URI=https://alertmanager.example.com ghcr.io/prymi
```

Note that latest main branch might have bugs or breaking changes. Using release images is strongly recommended for any production use.

Building a Docker image

```
make docker-image
```

This will build a Docker image locally from sources.

Health checks

`/health` endpoint can be used for health check probes, it always responds with `200 OK` code and `Pong` response body.

Configuration

Please see CONFIGURATION for full list of available configuration options and example.yaml for a config file example.

Contributing

Please see CONTRIBUTING for details.

License

Apache License 2.0, please see LICENSE.

---

karma is maintained by prymitive.
This page was generated by GitHub Pages.

Configuration options

Alert dashboard for Prometheus Alertmanager

View on GitHub

Configuration options

Config file

By default karma will try to read configuration file named `karma.yaml` from current directory. Configuration file uses YAML format and it needs to have `.yaml` extension. Custom filename and directory can be passed via command line flags or environment variables:

- `--config.file` flag or `CONFIG_FILE` env variable - path to the config file

Example with flags:

```
karma --config.file docs/example.yaml
```

Example with environment variables:

```
CONFIG_FILE="docs/example.yaml"
```

Authentication

`authentication` sections allows enabling authentication support in karma. When set users will be required to authenticate when trying to access karma. There are currently two supported authentication methods:

- Basic HTTP Authentication. Karma will be performing authentication using configured list of username & password pairs. This method is only recommended for testing.
- External authentication via headers. Karma doesn't perform any authentication itself, it is done by a frontend service (SSO or nginx reverse proxy) that sets a header with username on every request.

Only one method can be enabled in the config. Enabling authentication will also force silences to be created with usernames passed from credentials. Syntax:

```
authentication:
  header:
    name: string
    value_re: regex
    group_name: string
    group_value_re: regex
    group_value_separator: string
  basicAuth:
    users:
      - username: string
        password: string
```

- `authentication:users:header:name` - name of the header that will contain the username. If this header is missing from a request access will be forbidden. When set header authentication is used.
- `authentication:users:header:value_re` - regex used to extract the username from the request header value (when `authentication:users:header:name` is set). It must include one numbered capturing group, whatever is matched by that group will be used as the silence form author field. All regexes are anchored. This option must be set when `authentication:users:header:name` is set.
- `authentication:users:header:group_name` - name of the header that will contain any groups the user has.
- `authentication:users:header:group_value_re` - Similar to `authentication:users:header:value_re`, but for groups instead of usernames. Must be set when `authentication:users:header:group_name` is set.

- `authentication:users:header:group_value_separator` - This will be used to split the group header to multiple group names. The split is done after evaluating the value regex. Default value is `" "`.
- `authentication:users` - list of users (username & password) allowed to login. Passwords are stored plain without any encryption. When set HTTP basic authentication will be used.

Defaults:

```
authentication:
  header:
    name: ""
    value_re: ""
  basicAuth:
    users: []
```

Example where HTTP Basic Authentication will be used with a list of username and password pairs set in karma config file.

```
authentication:
  basicAuth:
    users:
      - username: alice
        password: secret
      - username: bob
        password: moreSecret
```

Example where the `X-Auth` header will be used for authentication, raw header value will be used as username.

```
authentication:
  header:
    name: X-Auth
    value_re: ^(.+)$
```

Example where the `X-Auth-User` and `X-Auth-Groups` headers will be used to set username and list of groups. This assume that `X-Auth-Groups` value has `Groups: foo,bar` syntax, where `foo` and `bar` are two groups user belongs to.

```
authentication:
  header:
    name: X-Auth-User
    value_re: ^(.+)$
    group_name: X-Auth-Groups
    group_value_re: 'Groups: (.+)'
    group_value_separator: ','
```

Authorization

`authorization` section allows to configure authorization groups used in silence ACL rules. Syntax:

```
authorization:
  acl:
    silences: string
  groups:
    - name: string
      members: list of strings
```

- `acl:silences` - path to silence ACL configuration file, see ACLs for details
- `groups` - list of group definitons, each group must have a `name` and `members` list. `name` will be used in silence ACL rules, `members` list should contain list of user names as passed from authentication layer.

Example with two groups using basic auth users and silences ACL config:

```
authentication:
  basicAuth:
    users:
      - username: alice
```

```
        password: secret
      - username: bob
        password: secret
      - username: john
        password: secret
authorization:
  acl:
    silences: /etc/karma/acls.yaml
  groups:
    - name: admins
      members:
        - alice
        - bob
    - name: users
      members:
        - john
```

Alertmanagers

`alertmanager` section allows setting Alertmanager servers that should be queried for alerts. You can configure one or more Alertmanager servers, alerts with identical label set will be deduplicated and labeled with each Alertmanager server they were observed at. This allows using karma to collect alerts from a pair of Alertmanager instances running in HA mode. Syntax:

```
alertmanager:
  interval: duration
  servers:
    - name: string
      cluster: string
      uri: string
      external_uri: string
      timeout: duration
      proxy: bool
      readonly: bool
      tls:
        ca: string
        cert: string
```

```
      key: string
      insecureSkipVerify: bool
    proxy_url: string
    headers:
      any: string
    cors:
      credentials: string
    healthcheck:
      visible: bool
      filters: map (string: list of strings)
```

- `interval` - how often alerts should be refreshed, a string in time.Duration format. If set to `1m` karma will query every Alertmanager server once a minute. This is global setting applied to every Alertmanager server. All instances will be queried in parallel. Note that the maximum value for this option is `15m`. The UI has a watchdog that tracks the timestamp of the last pull. If the UI does not receive updates for more than 15 minutes it will print an error and reload the page.
- `name` - name of this Alertmanager server, will be used as a label added to every alert in the UI and for filtering alerts using `@alertmanager=NAME` filter
- `cluster` - this option can be set to give Alertmanager clusters custom names in the UI. If there are multiple alertmanager servers configured in karma config that are part of the same HA cluster then this option should be set to the same value for all of them. If `cluster` option isn't set a name will be generated for each detected cluster.
- `uri` - base URI of this Alertmanager server. Supported URI schemes are `http://` and `https://`. If URI contains basic auth info (`https://user:password@alertmanager.example.com`) and you don't want it to be visible to users then ensure `proxy: true` is also set in order to avoid leaking auth information to the browser. Note: if URI contains username and password and proxy option is NOT enabled (see below), then the username & password information will be stripped from the URI and `Authorization` header using Basic Auth will be set for all in browser requests.
- `external_uri` - this option allows to override base URI of this Alertmanager used for browser links and also silence requests (but only when proxy mode is not enabled).

- `timeout` - timeout for requests send to this Alertmanager server, a string in time.Duration format.
- `proxy` - if enabled requests from user browsers to this Alertmanager will be proxied via karma. This applies to requests made when managing silences via karma (creating or expiring silences). This option cannot be used when `readonly` is enabled.
- `readonly` - set this Alertmanager upstream to a read only mode. This will disallow silence creation or editing. This option cannot be used when `proxy` is enabled.
- `tls:ca` - path to CA certificate used to establish TLS connection to this Alertmanager instance (for URIs using `https://` scheme). If unset or empty string is set then Go will try to find system CA certificates using well known paths.
- `tls:cert` - path to a TLS client certificate file to use when establishing TLS connections to this Alertmanager instance if it requires a TLS client authentication. Note that this option requires `tls:key` to be also set.
- `tls:key` - path to a TLS client key file to use when establishing TLS connections to this Alertmanager instance if it requires a TLS client authentication. Note that this option requires `tls:cert` to be also set.
- `tls:insecureSkipVerify` - disable server certificate validation, can be set to allow using self-signed certs, use at your own risk
- `proxy_url` - sets a proxy for HTTP client used for making requests to the upstream server. This can be used to access servers available via SOCKS5 proxy.
- `headers` - a map with a list of key: values which are header: value. These custom headers will be sent with every request to the alert manager instance. **NOTE**: these headers are only sent for alertmanager requests, they are NOT set on requests send to Prometheus server when querying alert history. Please see `history:rewrite` section below if you want to set headers for Prometheus requests.
- `cors:credentials` - sets the CORS credentials settings for browser requests, see docs for the list of possible values. By default credentials are included in all requests (`include`), set it to `omit` or `same-origin` if Alertmanager is configured to respond with `Access-Control-Allow-Origin: *` , see docs.

- `healthcheck:visible` - enable this option if you want `healthcheck:filters` alerts to be visible in karma UI. An alternative to enabling this option is to route healcheck alerts to alertmanager receiver that isn't visible using default karma filters.
- `healthcheck:filters` - define healtchecks using alert filters. When set karma will search for alerts matching defined filters and show an error if it doesn't match anything. This can be used with a Dead man's switch style alert to notify karma users that there's a problem with alerting pipeline. Syntax for this option is a map where key is the name of the filter set (used in the UI when showing errors) and the value is a list of filters.

  Example:

  - Setup always on alert in each Prometheus server (prom1 and prom2):

    ```
    - alert: DeadMansSwitch
      expr: vector(1)
    ```

  - Add healtcheck configuration to karma:

    ```
    alertmanager:
      servers:
        - name: am
          uri: https://alertmanager.example.com
          healthcheck:
            filters:
              prom1:
                - alertname=DeadMansSwitch
                - instance=prom1
              prom2:
                - alertname=DeadMansSwitch
                - instance=prom2
    ```

  If any of these alerts is missing from alertmanager karma will show a warning in the UI.

Note: there are multiple supported combination of URI settings which result in a slightly different behavior. Settings that control it are:

- `uri` - this option tells karma backend the URI that should be used to collect all alerts and silence data from given Alertmanager instance. This setting is required.
- `proxy` - this option when set to true enables karma backend to proxy all silence management requests (creating, editing or deleting silences via karma UI), so when the user creates a silence via karma UI the browser makes a request to karma backend, the backend then forwards this request to the Alertmanager using the value of `uri` option as the URI. When this option is set to `false` all browser requests will use `uri` value. This setting is optional, default value for it is `false`.
- `external_uri` - this option tells karma how the browser should connect to given Alertmanager instance, it can be used for silence management requests (creating, editing or deleting silences via karma UI) and how to generate links to silences in Alertmanager web UI. Behavior of this option depends on the value of `proxy` setting. When proxy mode is enabled:
  - silence management requests will use karma backend URI
  - silence links to Alertmanager web UI will use `external_uri` value as base URI When proxy mode is disabled:
  - silence management requests will use `external_uri` value as base URI
  - silence links to Alertmanager web UI will use `external_uri` value as base URI

Breakdown of all combination of settings:

1. Only `uri` is set:

   ```
   uri: http://localhost:123
   ```

   Karma would use those URIs for:

Backend	Silence management	Silence links
`http://localhost:123`	`http://localhost:123`	`http://localhost:123`

2. Proxy mode is enabled:

```
uri: http://localhost:123
proxy: true
```

Karma would use those URIs for:

Backend	Silence management	Silence links
`http://localhost:123`	Karma internal URI	`http://localhost:123`

3. `external_uri` is set, but proxy mode is disabled:

```
uri: http://localhost:123
external_uri: http://example.com
```

Karma would use those URIs for:

Backend	Silence management	Silence links
`http://localhost:123`	`http://example.com`	`http://example.com`

4. Proxy mode is enabled and `external_uri` is set:

```
uri: http://localhost:123
proxy: true
external_uri: http://example.com
```

Karma would use those URIs for:

Backend	Silence management	Silence links
`http://localhost:123`	Karma internal URI	`http://example.com`

5. ReadOnly mode is enabled:

```
uri: http://localhost:123
readonly: true
```

Karma would use those URIs for:

Backend	Silence management	Silence links
`http://localhost:123`	Disabled	`http://localhost:123`

Example with two production Alertmanager instances running in HA mode and a staging instance that is also proxied and requires a custom auth header:

```
alertmanager:
  interval: 1m
  servers:
    - name: production1
      uri: https://alertmanager1.prod.example.com
      timeout: 20s
      proxy: false
    - name: production2
      uri: https://alertmanager2.prod.example.com
      timeout: 20s
      proxy: false
    - name: staging
      uri: https://alertmanager.staging.example.com
      timeout: 30s
      proxy: true
      tls:
        ca: /etc/ssl/staging-ca.crt
      headers:
        X-Auth-Token: aValidToken
    - name: protected
      uri: https://alertmanager-auth.prod.example.com
```

```
      timeout: 20s
      tls:
        cert: /etc/ssl/client.pem
        key: /etc/ssl/client.key
    - name: self-signed
      uri: https://test.example.com
      tls:
        insecureSkipVerify: true
    - name: socks5
      uri: https://internal.address
      proxy_url: socks5://proxy.local:5000
```

Defaults:

```
alertmanager:
  interval: 1m
  servers: []
```

There is no default for `alertmanager.servers` and it's a required option for setting multiple Alertmanager servers. For cases where only a single server needs to be configured without a config file see Simplified Configuration.

Alert acknowledgement

Prometheus Alertmanager allows alerts to be in 3 states:

- `active` - when alert is firing
- `suppressed` - when alert is either silenced by a silence rule or inhibited by another alert using inhibition rules
- `unprocessed` - initial state for new alerts before they are checked against all silence rules so Alertmanager doesn't yet know if the alert should be `active` or `suppressed`

A silence rule can be used to mark an alert as acknowledged and being worked on. To simplify creating of such silences karma provides a one click button that will create a silence matching alert group it was clicked for. `alertAcknowledgement` allows to enable this feature and customize it's configuration. Syntax:

```
alertAcknowledgement:
  enabled: bool
  duration: duration
  author: string
  comment: string
```

- `enabled` - setting it to true will enable creation of short lived acknowledgement silences.
- `duration` - duration for acknowledgement silences, value is a string in time.Duration format.
- `author` - default author for acknowledgement silences. If user set the author field on the silence form then that value will be used instead.
- `comment` - custom comment used for acknowledgement silences (optional). If the comment contains `%NOW%` it will be replaced by current timestamp with UTC timezone, to use timestamp with local timezone use `%NOWLOC%`.

Defaults:

```
alertAcknowledgement:
  enabled: false
  duration: 15m0s
  author: karma
  comment: ACK! This alert was acknowledged using karma on %NOW%
```

A common problem is setting a correct duration for the silence. If set for too short it can expire before the issue is resolved, and will require re-silencing all the alerts. If set for too long it mask the same problem reoccurring in the future. This requires user to expire the silence once the issue is resolved.

kthxbye is a tiny daemon that can help with managing short lived acknowledged silences. It will continuously extend short lived acknowledgement silences if there are alerts firing against those silences, which means that the user doesn't need to worry about setting proper duration for such silences. To use it run an instance of kthxbye with every alertmanager instance or cluster and configure it to use the same comment prefix in `comment`. With this setup when user clicks to acknowledge an alert karma will create a short lived silence and kthxbye will keep that silence in

Alertmanager until there are no alerts matching it, meaning that the issue was resolved.

Annotations

`annotations` section allows configuring how alert annotation are displayed in the UI. Syntax:

```
annotations:
  default:
    hidden: bool
  hidden: list of strings
  visible: list of strings
  keep: list of strings
  strip: list of strings
  order: list of strings
  actions: list of strings
  enableInsecureHTML: bool
```

- `default:hidden` - bool, true if all annotations should be hidden by default.
- `hidden` - list of annotations that should be hidden by default.
- `visible` - list of annotations that should be visible by default when `default:hidden` is set to `true`.
- `keep` - list of allowed annotations, if empty all annotations are allowed.
- `strip` - list of ignored annotations.
- `order` - custom order of annotation names. All annotations listed here will appear first in the order specified here. Remaining annotations will be sorted alphabetically and appended at the end.
- `actions` - list of annotations that will be moved to alert dropdown menu. this only applies to annotations where value is a link.
- `enableInsecureHTML` - by default all annotation values are escaped when rendered in users browser, to prevent any injection attacks. If this option is set to `true` escaping will be disabled which allows HTML tags to be used in annotations, but if someone manages to send alerts with annotations containing untrusted HTML/Javascript code to your alertmanager instances karma will allow it to be executed in your browser.

NOTE Enable at your own risk.

The difference between `hidden` / `visible` and `keep` / `strip` is that hidden annotations are still accessible, but they are shown in the UI collapsed by default (only name is visible, value is shown after clicking), while stripped annotations are removed entirely and never presented to the user.

Example where all annotations except `summary` are hidden by default. If there are additional annotation keys user will need to click on the `+` icon to see them. `summary` annotation will always appear first in the UI, followed by `help` and all other annotations (sorted alphabetically). Any annotation with name `jira` and a value that is a URL will be moved to alerts dropdown menu.

```
annotations:
  default:
    hidden: true
  hidden: []
  visible:
    - summary
  keep: []
  strip:
    - help
    - verylong
  order:
    - summary
    - help
  actions:
    - jira
```

Example where all annotations except `details` are visible by default. If `details` annotation is present on any alert user will need to click on the `+` icon to see it. Additionally `secret` annotation is stripped and never shown in the UI.

```
annotations:
  default:
    hidden: false
  hidden:
    - details
```

```
  visible: []
  keep: []
  strip:
    - secret
```

Defaults:

```
annotations:
  default:
    hidden: false
  hidden: []
  visible: []
  keep: []
  strip: []
  order: []
  actions: []
  enableInsecureHTML: false
```

Filters

`filters` section allows configuring default set of filters used in the UI.

Syntax:

```
filters:
  default: list of strings
```

- `default` - list of filters to use by default when user navigates to karma web UI. Visit `/help` page in karma for details on available filters. Note that if a string starts with `@` YAML requires to wrap it in quotes.

Example:

```
filters:
  default:
```

```
    - "@state=active"
    - severity=critical
```

Defaults:

```
filters:
  default: []
```

Grid

`grid` section allows customizing how alert grid is rendered in the UI. Sorting configuration can be overridden by each user via UI settings. Syntax:

```
grid:
  sorting:
    order: string
    reverse: bool
    label: string
    customValues:
      labels: dict
  auto:
    ignore: list of strings
    order: list of strings
  groupLimit: integer
```

- `sorting:order` - default sort order for alert grid, valid values are:
  - `disabled` - no sorting, alert groups are rendered in the order they are returned by the API
  - `startsAt` - sort by alert timestamps, most recent alert in each group will be used when comparing each group
  - `label` - sort by labels, if the label used for sorting is not shared by all alerts in a group then the first alert in the group will be queried for it
- `sorting:reverse` - default value for reversed sort order
- `sorting:label` - label name for sorting when `grid:sorting:order` is set to `label`. Labels can be assigned custom values used only by sorting via `sorting:customValues:labels`.

- `sorting:customValues:labels` - when sorting using alert labels values are compared as strings, which work for labels like `cluster=A`, `cluster=B` & `cluster=C`, but not for `cluster=prod`, `cluster=staging` & `cluster=dev`. Alphabetic sort would order the second case as follows: `dev`, `prod`, `staging`. To allow for more natural sorting `sorting:valueMapping` can be used to map label values to integer values which will be used for sorting instead of original string values. Note: this option is not available via environment variables, you can only set it via the config file.
- `auto:ignore` - list of label names that should never be selected as multi-grid source label when multi-grid is configured to `Automatic selection` in the UI or when `ui:multiGridLabel` is set to `@auto`.
- `auto:order` - preferred order for selecting labels to be used as multi-grid source label when multi-grid is configured to `Automatic selection` in the UI or when `ui:multiGridLabel` is set to `@auto`. If a label name is not present in this list labels with equal weight will be picked in alphabetic order.
- `groupLimit` - default number of alert groups to show in the UI, loading more will require user to click on `Load more` button.

Defaults:

```
grid:
  sorting:
    order: startsAt
    reverse: true
    label: alertname
    customValues:
      labels: {}
  auto:
    ignore: []
    order: []
  groupLimit: 40
```

Example with sorting using `severity` label and value mappings for it:

```
grid:
  sorting:
    order: label
```

```
    reverse: false
    label: severity
    customValues:
      labels:
        severity:
          critical: 1
          warning: 2
          info: 3
```

Alert history

`history` section allows to enable and configure alert history queries. When enabled karma will use `source` fields to try finding remote Prometheus servers sending alerts. If `source` is a link that points at a reachable Prometheus server then karma will query its metrics to estimate how many times did that alert fire in the last 24h.

Syntax:

```
history:
  enabled: bool
  timeout: duration
  workers: integer
  rewrite:
    - source: regex
      uri: string
      proxy_url: string
      headers:
        any: string
      tls:
        ca: string
        cert: string
        key: string
        insecureSkipVerify: bool
```

- `enabled` - enable alert history UI and backend query support
- `timeout` - timeout for HTTP requests send to remote Prometheus servers

- `workers` - number of worker threads to start, each worker handles one outgoing HTTP request, more workers allows to handle more concurrent queries if you have a large number of Prometheus servers sending alerts
- `rewrite` - list of source rewrite rules applied before any request is send to remote Prometheus. Rewrite rules can be used to modify URI or TLS settings used by karma when connecting to Prometheus API if `source` field in alert uses addresses not reachable from karma. All regexes are anchored, `${N}` syntax can be used for capture groups. You can rewrite uri to an empty string to disable connecting to that specific Prometheus instance.

Defaults:

```
history:
  enabled: true
  timeout: 20s
  workers: 30
  rewrite: []
```

Example with rewrite rule that will replace `https://prometheus.example.com` with `http://localhost:9093`:

```
history:
  rewrite:
    - source: 'https://prometheus.example.com'
      uri: 'http://localhost:9093'
```

Example with rewrite rule that will replace `https://*.example.com` with `http://prometheus-*.internal` (`https://dev.example.com` becomes `http://prometheus-dev.example.com`):

```
history:
  rewrite:
    - source: 'https://(.+).example.com'
      uri: 'http://prometheus-$1.internal'
```

Example with rewrite rule that will disable sending any history queries to `http://prometheus.internal` :

```
history:
  rewrite:
    - source: 'http://prometheus.internal'
      uri: ''
```

Example with rewrite rule that configures TLS settings without modifying URI:

```
history:
  rewrite:
    - source: '(.*)'
      uri: '$1'
      tls:
        insecureSkipVerify: true
```

Example with rewrite rule that configures a proxy without modifying URI:

```
history:
  rewrite:
    - source: '(.*)'
      uri: '$1'
      proxy_url: socks5://proxy.local:5000
```

Example with rewrite rule that will set an extra header for all history request send to Prometheus server `http://prometheus.example.com` :

```
history:
  rewrite:
    - source: 'http://prometheus.example.com'
      headers:
        X-Auth: secret
        X-Foo: bar
```

Karma

`karma` section allows configuring miscellaneous internal options.

Syntax:

```
karma:
  name: string
```

- `name` - name of given karma instance, this is currently used for the browser tab title.

Defaults:

```
karma:
  name: karma
```

Labels

`labels` section allows configuring how alert labels will be rendered in the UI. All labels will be parsed when collecting alerts from Alertmanager API and used when deduplicating alerts, but some labels aren't useful to users and so can be removed from the UI, this is controlled by `keep`, `keep_re`, `strip` and `strip_re` options. `colors` section allows configuring which labels should have colors applied to label background in the UI. Colors can help visually identify alerts with shared labels, for example coloring hostname label will allow to quickly spot all alerts for the same host. Syntax:

```
labels:
  color:
    static: list of strings
    unique: list of strings
    custom:
      foo:
        - value: string
          value_re: regex
```

```
        color: string
order: list of strings
keep: list of strings
keep_re: list of regex
strip: list of strings
strip_re: list of regex
valueOnly: list of strings
valueOnly_re: list of regex
```

- `color:static` - list of label names that will all have the same color applied (different than the default label color). This allows to quickly spot a specific label that can have high range of values, but it's important when reading the dashboard. For example coloring the instance label allows to quickly learn which instance is affected by given alert.
- `color:unique` - list of label names that should have unique colors generated in the UI.
- `color:custom` - nested map of label names and value with colors - this allows to configure a set of labels with custom predefined colors applied to them rather than generated. Value is a mapping with `label name -> list of dicts`, each dict object allows setting:
  - `value` - the exact value of the label to match against
  - `value_re` - Go compatible regular expression to match against. All regexes will be automatically anchored.
  - `color` : color to apply if either `value` or `value_re` matches

  Either `value` or `value_re` is required, both can be set in which case `value` with be tested first. Entries are evaluated in the order they appear in the config file. Note: this option is not available via environment variables, you can only set it via the config file.

- `order` - custom order of label names. All labels listed here will appear first in the order specified here. Remaining labels will be sorted alphabetically and appended at the end.
- `keep` - list of allowed labels, if both `keep` and `keep_re` are empty all labels are allowed.

- `keep_re` - list of Go compatible regular expressions to keep matching labels; all regexes will be automatically anchored; if both `keep` and `keep_re` are empty all labels are allowed.
- `strip` - list of ignored labels.
- `strip_re` - list of Go compatible regular expressions to ignore matching labels; all regexes will be automatically anchored.
- `valueOnly` - list of label names for which only the value will be displayed in the UI.
- `valueOnly_re` - list of JavaScript compatible regular expressions to display only the value for matching labels; all regexes will be automatically anchored.

Example with static color for the `job` label (every `job` label will have the same color regardless of the value) and unique color for the `@receiver` label (every `@receiver` label will have color unique for each value).

```
labels:
  color:
    static:
      - job
    unique:
      - "@receiver"
```

Example where `task_id` label is ignored by karma:

```
labels:
  keep: []
  strip:
    - task_id
```

Example where all but `instance` and `alertname` labels are allowed:

```
labels:
  keep:
    - alertname
    - instance
  strip: []
```

Example where only labels with the prefix `custom_` are allowed:

```
labels:
  keep: []
  keep_re:
    - 'custom_.*'
```

Example where `severity` label will have a red color for `critical`, yellow for `warning` and blue for `info`:

```
labels:
  color:
    custom:
      "@alertmanager":
        - value: prod
          color: "#e6e"
      severity:
        - value: info
          color: "#87c4e0"
        - value: warning
          color: "#ffae42"
        - value: critical
          color: "#ff220c"
```

Example with a regex value, `info`, `warning` and `critical` will get colors as below, but any value not matching those 3 values will use the color from `.*`:

```
labels:
  color:
    custom:
      severity:
        - value: info
          color: "#87c4e0"
        - value: warning
          color: "#ffae42"
        - value: critical
          color: "#ff220c"
```

```
        - value_re: ".*"
          color: "#736598"
```

Note: be sure to set fallback values at the end of the list, so they're only evaluated if there's no exact value match

Defaults:

```
labels:
  color:
    static: []
    unique: []
    custom: {}
  keep: []
  keep_re: []
  strip: []
  strip_re: []
  valueOnly: []
  valueOnly_re: []
```

Listen

`listen` section allows configuring karma web server behavior. Syntax:

```
listen:
  address: string
  port: integer
  timeout:
    read: duration
    write: duration
  prefix: string
  tls:
    cert: string
    key: string
  cors:
    allowedOrigins: list of strings
```

- `address` - Hostname or IP to listen on.
- `port` - HTTP port to listen on.
- `timeout:read` - HTTP server request read timeout
- `timeout:write` - HTTP server response write timeout
- `prefix` - URL root for karma, you can use to if you wish to serve it from location other than `/`. This option is mostly useful when using karma behind reverse proxy with other services on the same IP but different URL root.
- `tls:cert` - path to a TLS certificate, enables listening on HTTPS instead of HTTP,
- `tls:key` - path to a TLS key, required when `tls.cert` is set
- `cors:allowedOrigins` - List of origins a cross-domain request can be executed from. An empty list means all origins are allowed.

Example where karma would listen for HTTP requests on `http://1.2.3.4:80/karma/`

```
listen:
  address: 1.2.3.4
  port: 80
  prefix: /karma/
```

Example where karma would listen for HTTPS requests on `https://1.2.3.4:443/`

```
listen:
  address: 1.2.3.4
  port: 443
  tls:
    cert: server.pem
    key: server.key
```

Defaults:

```
listen:
  address: "0.0.0.0"
  port: 8080
  prefix: /
  tls:
```

```
    cert: ""
    key: ""
  cors:
    allowedOrigins: []
```

Log

`log` section allows configuring logging subsystem. Syntax:

```
log:
  config: bool
  level: string
  format: string
  requests: bool
  timestamp: bool
```

- `config` - if set to `true` karma will log used configuration on startup
- `level` - log level to set for karma, possible values are `debug`, `info`, `warning`, `error`, `fatal` and `panic`.
- `format` - controls how log messages are formatted, possible values are `text` and `json`. If set to `json` each log will be a JSON object
- `requests` - if set to `true` karma will log all requests
- `timestamp` - if set to `true` all log messages will include a timestamp

Defaults:

```
log:
  config: false
  level: info
  format: text
  requests: false
  timestamp: false
```

Silences

`silences` section allows specifying to configure silence post post-processing. Syntax:

```
silences:
  expired: duration
  comments:
    linkDetect:
      rules: list of link detection rules
```

- `expired` - controls how long expired silences are shown on active alerts. If `expired` is set to `5m` silences expired in the last 5 minutes will be shown. Set it to zero or a negative value to disable showing expired silences.
- `comments:linkDetect:rules` - allows to specify a list of rules to detect links inside silence comments. It's intended to find ticket system ID strings and turn them into links. Each rule must specify:
  - `regex` - regular expression that matches ticket system IDs. Each regex must contain at least one capture group `(regex)`. All regexes will be automatically anchored.
  - `uriTemplate` - template string that will be used to generate a link. Each template must include `$1` which will be replaced with text matched by the `regex`.

Examples where alerts that got unsilenced will show silences expired in the last 15 minutes:

```
silences:
  expired: 15m
```

Examples where alerts that got unsilenced will not show recently expired silences:

```
silences:
  expired: -1m
```

Example where a string `DEVOPS-123` inside a comment would be rendered as a link to a JIRA ticket `https://jira.example.com/browse/DEVOPS-123`.

```
silences:
  comments:
    linkDetect:
      rules:
        - regex: "(DEVOPS-[0-9]+)"
          uriTemplate: https://jira.example.com/browse/$1
```

Receivers

`receivers` section allows configuring how alerts from different receivers are handled by karma. If alerts are routed to multiple receivers they can be duplicated in the UI, each instance will have different value for `@receiver`. Syntax:

```
receivers:
  keep: list of strings
  strip: list of strings
```

- `keep` - list of receivers name that are allowed, if empty all receivers are allowed.
- `strip` - list of receiver names that will not be shown in the UI.

Example where alerts that are routed to the `alertmanage2es` receiver are ignored by karma.

```
receivers:
  strip:
    - alertmanage2es
```

Defaults:

```
receivers:
  strip: []
```

Silence form

`silenceForm` section allows customizing silence form behavior.

Syntax:

```
silenceForm:
  defaultAlertmanagers: list of strings
  strip:
    labels: list of strings
```

- `defaultAlertmanagers` - list of Alertmanager names that will be used as default when creating a new silence. If selected alertmanager is part of a cluster then the whole cluster will be used in the silence form.
- `strip:labels` - list of labels to ignore when populating silence form from individual alerts or group of alerts. This allows to create silences matching only unique labels, like `instance` or `host`, ignoring any common labels like `job`.

Example where `job` label won't be auto populated in the silence form.

```
silenceForm:
  strip:
    labels:
      - job
```

Example where alertmanagers `prod1` and `prod2` will be the default ones when creating a new silence

```
silenceForm:
  defaultAlertmanagers:
    - prod1
    - prod2
```

UI defaults

`ui` section allows configuring default values for UI settings controled via the configuration modal. Those defaults can be overwritten by use via UI controls.

Syntax:

```
ui:
  refresh: duration
  hideFiltersWhenIdle: bool
  colorTitlebar: bool
  theme: string
  animations: bool
  minimalGroupWidth: integer
  alertsPerGroup: integer
  collapseGroups: string
  multiGridLabel: string
  multiGridSortReverse: bool
```

- `refresh` - default refresh interval, this tells the UI how often karma API should be queried
- `hideFiltersWhenIdle` - if enabled filter bar will be hidden after some user inactivity
- `colorTitlebar` - if enabled alert group title bar color will be set to follow alerts in that group
- `theme` - default theme, possible values:
  - `light` - bright theme
  - `dark` - dark theme
  - `auto` - follows browser preferences using prefers-color-scheme media queries

  Default value is `auto`.
- `animations` - enables UI animations
- `minimalGroupWidth` - minimal width (in pixels) for each alert group rendered on the grid. This value is used to calculate the number of columns rendered on the grid.
- `alertsPerGroup` - default number of alerts to show for each group
- `collapseGroups` - controls if alert groups will default to being rendered expanded or collapsed (only title bar is visible). Valid values:
  - expanded - groups are always expanded

    - collapsed - groups are always collapsed
    - collapsedOnMobile - groups are expanded on desktop and collapsed on mobile browsers
- `multiGridLabel` - when set to a label name it enables multi-grid support. With multi-grid karma will have a dedicated grid for each value of this label, all alerts sharing that value will be placed on the same grid. There will be extra grid for alerts without that label. Grid sorting options will be used to sort the list of grids. This option accepts additional special values:
    - `@auto` - grid label will be selected automatically
    - `@alertmanager` - one grid per alertmanager configured in karma config
    - `@cluster` - one grid per alertmanager cluster
    - `@receiver` - one grid per alertmanager receiver
- `multiGridSortReverse` - when multi-grid is enabled set to `true` the order in which grids are displayed.

Defaults:

```
ui:
  refresh: 30s
  hideFiltersWhenIdle: true
  colorTitlebar: false
  theme: "auto"
  animations: true
  minimalGroupWidth: 420
  alertsPerGroup: 5
  collapseGroups: collapsedOnMobile
  multiGridLabel: ""
  multiGridSortReverse: false
```

Customizing karma

In order to keep the core code simple karma doesn't support any way of extending provided functionality. There is however possibility to inject custom CSS & JavaScript code, which can be used to either override built in CSS styles or integrate with extra services.

```
custom:
  css: string
  js: string
```

- `css` - path to a CSS file
- `js` - path to JavaScript file

Example:

```
custom:
  css: /theme/custom.css
  js: /assets/custom.js
```

Use at your own risk and be aware that used CSS class names might change without warning. This feature is provided as is without any guarantees.

Command line flags

Config file options are mapped to command line flags, so `alertmanager:interval` config file key is accessible as `--alertmanager.interval` flag, run `karma --help` to see a full list. Exceptions for passing flags:

- `jira` - this option is a list of maps and it's only available when using config file.

There's no support for configuring multiple Alertmanager servers using flags, but it's possible to configure a single Alertmanager instance this way, see the Simplified Configuration section.

Environment variables

Environment variables are mapped in a similar way as command line flags, `alertmanager:interval` is accessible as `ALERTMANAGER_INTERVAL` env. Exceptions for passing flags:

- `HOST` - used by gin webserver, same effect as setting `listen:address` config option
- `PORT` - used by gin webserver, same effect as setting `listen:port` config option

There's no support for configuring multiple alertmanager servers using environment variables, but it's possible to configure a single Alertmanager instance this way, see the Simplified Configuration section.

Simplified Configuration

To configure multiple Alertmanager instances karma requires a config file, but for a single Alertmanager instance cases it's possible to configure all Alertmanager server options that are set for `alertmanager.servers` config section using only flags or environment variables.

Alertmanager URI

To set the `uri` key from `alertmanager.servers` map `ALERTMANAGER_URI` env or `--alertmanager.uri` flag can be used. Examples:

```
ALERTMANAGER_URI=https://alertmanager.example.com karma
karma --alertmanager.uri https://alertmanager.example.com
```

Alertmanager external URI

To set the `external_uri` key from `alertmanager.servers` map `ALERTMANAGER_EXTERNAL_URI` env or `--alertmanager.external_uri` flag can be used. Examples:

```
ALERTMANAGER_EXTERNAL_URI=https://alertmanager.example.com karma
karma --alertmanager.external_uri https://alertmanager.example.com
```

Alertmanager name

To set the `name` key from `alertmanager.servers` map `ALERTMANAGER_NAME` env or `--alertmanager.name` flag can be used. Examples:

```
ALERTMANAGER_NAME=single karma
karma --alertmanager.name single
```

Alertmanager timeout

To set the `timeout` key from `alertmanager.servers` map `ALERTMANAGER_TIMEOUT` env or `--alertmanager.timeout` flag can be used. Examples:

```
ALERTMANAGER_TIMEOUT=10s karma
karma --alertmanager.timeout 10s
```

Alertmanager request proxy

To set the `proxy` key from `alertmanager.servers` map `ALERTMANAGER_PROXY` env or `--alertmanager.proxy` flag can be used. Examples:

```
ALERTMANAGER_PROXY=true karma
karma --alertmanager.proxy
```

Silence Access Control Lists

Alert dashboard for Prometheus Alertmanager

View on GitHub

Silence Access Control Lists

Intro

Karma provides ability to setup ACLs for silences created by users. This can be used to limit what kind of silences each user is allowed to create, which can help to avoid, for example, `Team A` accidentally silencing alerts for `Team B`, or blocking engineering team from creating any silence at all, leaving that ability only to the sys admin / SRE team.

Example Alertmanager silence:

```
{
  "matchers": [
    {
      "name": "alertname",
      "value": "Test Alert",
      "isRegex": false,
      "isEqual": true
    },
    {
      "name": "cluster",
      "value": "prod",
      "isRegex": false,
      "isEqual": true
    },
    {
      "name": "instance",
      "value": "server1",
```

```
      "isRegex": false,
      "isEqual": true
    }
  ],
  "startsAt": "2020-03-09T20:11:00.000Z",
  "endsAt": "2020-03-09T21:11:00.000Z",
  "createdBy": "me@example.com",
  "comment": "Silence Test Alert on server1"
}
```

It would be applied to all alerts with name `Test Alert` and where label `cluster` is equal to `prod`. An ACL rule could be used to restrict silence creation based on matched labels, so for example only selected users would be allowed to silence this specific alert.

Requirements

For ACLs to work a few configuring options are required:

- `authorization:acl:silences` is set with acl config file path, there's no support for configuring ACLs via environment variables
- `proxy` must be enabled in karma configuration for each Alertmanager server where ACLs will be applied. `proxy: true` tells karma UI to proxy all silence operation requests (creating, editing & deleting silences) via karma backend. Since ACLs are applied in the proxy code it needs to be enabled to take effect. It is recommended to block ability for users to connect directly to Alertmanager servers to avoid bypassing ACL rules (alertmanager accepts all silences).

Optional configuration:

- `authentication` if configured user based matching of ACLs can be used, Header authentication with a frontend authentication proxy that passes usernames via header is recommended. This can be done with nginx configured as an authentication reverse proxy or proxy services like Cloudflare Access.

- `authorization:groups` must be configured if group polices will be used. This configuration maps users into groups, allowing to use those groups in ACL rules.

Regex silences

Alertmanager silences allow to use regex rules which can make it tricky to apply ACLs to those silences.

Silence example using regex:

```
{
  "matchers": [
    {
      "name": "alertname",
      "value": "Test Alert",
      "isRegex": false,
      "isEqual": true
    },
    {
      "name": "cluster",
      "value": "staging|prod",
      "isRegex": true,
      "isEqual": true
    }
  ],
  "startsAt": "2020-03-09T20:11:00.000Z",
  "endsAt": "2020-03-09T21:11:00.000Z",
  "createdBy": "me@example.com",
  "comment": "Silence Test Alert in staging & prod cluster"
}
```

The difference compared to the previous example is that the `cluster` label is now matched using `staging|prod` regex, so any alert with `cluster` label equal to `staging` or `prod` will be matched. This is a simple example, regexes allow to create very complex matching rules.

The effect on ACL rules can be illustrated with this example: let's say we have a group that should never be allowed to create any silence for `prod` cluster, so a silence like the one below should be blocked:

```
{
  "matchers": [
    {
      "name": "alertname",
      "value": "Test Alert",
      "isRegex": false,
      "isEqual": true
    },
    {
      "name": "cluster",
      "value": "prod",
      "isRegex": false,
      "isEqual": true
    }
  ],
  "startsAt": "2020-03-09T20:11:00.000Z",
  "endsAt": "2020-03-09T21:11:00.000Z",
  "createdBy": "me@example.com",
  "comment": "Silence Test Alert in prod cluster"
}
```

But if we would create an ACL rule that simply blocks silences with matcher:

```
{
  "name": "cluster",
  "value": "prod",
  "isRegex": false,
  "isEqual": true
}
```

then any user could bypass that with a regex matcher like:

```
{
  "name": "cluster",
```

```
  "value": "pro[d]",
  "isRegex": true,
  "isEqual": true
}
```

Because of that it is *highly recommended* to block regex silences, which can be done with an ACL rule. Since rules are evaluated in the order they are listed in the config file it is best to set this as the very first rule. See examples below to learn how to block regex silences.

Configuration syntax

- `rules` - list of silence ACL rules, rules are evaluated in the order they appear in this list

Rule syntax:

```
action: string
reason: string
scope:
  groups: list of strings
  alertmanagers: list of strings
  filters: list of filters
matchers:
  required: list of silence matchers
```

- `action` - this is the name of the action to take if given ACL matches all the conditions. Valid actions are:
  - `allow` - skip all other ACLs and allow silence to be created
  - `block` - skip all other ACLs and block silences from being created
  - `requireMatcher` - block silence if it doesn't have all of matchers specified in `matchers:required`
- `reason` - message that will be returned to the user if this ACL blocks any silence
- `scope` - this section contains all conditions required to apply given ACL rule to specific silence, if it's skipped then ACL rule will be applied to all users and

every silence

- `scope:groups` - list of group names from `authorization:groups`, if no group is specified here then this ACL will be applied to all users
- `scope:alertmanagers` - list of alertmanager names as specified in `alertmanager:servers`, if no name is specified here then this ACL will be applied to silences for all alertmanager servers
- `scope:filters` - list of matcher filters evaluated when checking if this ACL should be applied to given silence. Those filters can be used to enforce ACL rules only to some silences and are compared against silence matchers. All filters must be matching for given silence for ACL rule to be applied. Syntax:

  ```
  name: string
  name_re: regex
  value: string
  value_re: regex
  isRegex: bool
  isEqual: bool
  ```

  Every rule must have `name` or `name_re` AND `value` or `value_re`.

  Filter works by comparing:

  - `name` and `name_re` with silence matcher `name`.
  - `value` and `value_re` with silence matcher `value`.
  - `isRegex` on the filter with `isRegex` on silence matcher, if `isRegex` is not set on a filter then that filter will match silences with both `true` and `false` value on silence `isRegex`.
  - `isEqual` on the filter with `isEqual` on silence matcher, if `isEqual` is not set on a filter then that filter will match silences with both `true` and `false` value on a silence `isEqual`.

  See examples below. All regexes will be automatically anchored.

- `matchers:required` - list of additional matchers that must be part of the silence if it matches groups, alertmanagers and filters. This is only used if `action` is set to `requireMatcher`. All regexes will be automatically anchored. Syntax for each `requireMatcher` entry:

```
name: string
name_re: regex
value: string
value_re: regex
isRegex: bool
isEqual: bool
```

Fields:

- `name` - name to match, silence will be required to have a matcher with this exact name.
- `name_re` - name regex to match against, silence will be required to have a matcher with `name` field that matches this regex.
- `value` - value to match, silence will be required to have a matcher with this exact value.
- `value_re` - value regex to match against, silence will be required to have a matcher with `value` field that matches this regex.
- `isRegex` - value of silence matcher `isRegex`, if not set on a required matcher then any value of `isRegex` on a silence will be allowed.
- `isEqual` - value of silence matcher `isEqual`, if not set on a required matcher then any value of `isEqual` on a silence will be allowed.

A single entry cannot have both `name` & `name_re` or `value` & `value_re` set at the same time.

Examples

Block all silences

This rule will match all silence and block it.

```
rules:
  - action: block
    reason: silences are blocked
    scope:
```

```
    filters:
      - name_re: .+
        value_re: .+
```

Block silences using regex matchers

This rule will match all silence with any matcher using regexes (`isRegex: true` on the matcher) and block it.

```
rules:
  - action: block
    reason: all regex silences are blocked, use only concrete label names and
    scope:
      filters:
        - name_re: .+
          value_re: .+
          isRegex: true
```

Block negative matchers on silences

This rule will match all silence with `isEqual: false` and block it.

```
rules:
  - action: block
    reason: silences are blocked
    scope:
      filters:
        - name_re: .+
          value_re: .+
          isEqual: false
```

Allow admin group to create any silence

```
rules:
  - action: allow
    reason: admins are allowed
    scope:
      groups:
        - admins
```

Allow only admins group to create silences with cluster=prod

First allow all members of the `admins` group to create any silence, then block silences with `cluster=prod`. Since ACL rules are evaluated in the order specified and first `allow` or `block` rule stops other rule processing this will allow `admins` to create `cluster=prod` silences while everyone else is blocked from it. Disabling regex rules as first steps prevents users from bypassing those ACLs with regex silences.

```
rules:
  - action: block
    reason: all regex silences are blocked, use only concrete label names and
    scope:
      filters:
        - name_re: .+
          value_re: .+
          isRegex: true
  - action: allow
    reason: admins are allowed
    scope:
      groups:
        - admins
  - action: block
    reason: only admins can create silences with cluster=prod
    scope:
      filters:
        - name: cluster
          value: prod
          isEqual: true
```

Require postgresAdmins group to always specify db=postgres in silences

Block `postgresAdmins` members from creating silences unless they add `db=postgres` to the list of matchers.

```
rules:
  - action: requireMatcher
    reason: postgres admins must add db=postgres to all silences
    scope:
      groups:
        - postgresAdmins
    matchers:
      required:
        - name: db
          value: postgres
          isEqual: true
```

Require devTeam group to specify instance=server1-3

Block devTeam members from creating silences unless they target one of the servers they own.

```
rules:
  - action: requireMatcher
    reason: devTeam can only silence owned servers
    scope:
      groups:
        - devTeam
    matchers:
      required:
        - name: instance
          value_re: server[1-3]
          isEqual: true
```

Require everyone to always specify `team` matcher in silences

Block anyone from creating silences unless they add `team` matcher with some value.

```
rules:
  - action: requireMatcher
    reason: team label is required for all silences
    matchers:
      required:
        - name: team
          value_re: .+
```

---

karma is maintained by prymitive.
This page was generated by GitHub Pages.

```yaml
alertmanager:
  interval: 60s
  servers:
    - name: local
      uri: http://localhost:9093
      timeout: 10s
      proxy: true
      readonly: false
      headers:
        X-Auth-Test: some-token-or-other-string
    - name: client-auth
      uri: https://localhost:9093
      timeout: 10s
      tls:
        ca: /etc/ssl/certs/ca-bundle.crt
        cert: /etc/karma/client.pem
        key: /etc/karma/client.key
annotations:
  default:
    hidden: false
  hidden:
    - help
  visible: []
custom:
  css: /custom.css
  js: /custom.js
debug: false
filters:
  default:
    - "@receiver=by-cluster-service"
karma:
  name: karma-prod
labels:
  color:
    static:
      - job
    unique:
      - cluster
      - instance
      - "@receiver"
  keep: []
  strip: []
listen:
  address: "0.0.0.0"
  port: 8080
  prefix: /
  cors:
    allowedOrigins:
      - https://example.com
log:
  config: false
  level: info
silences:
  comments:
    linkDetect:
      rules:
```

```
        - regex: "(DEVOPS-[0-9]+)"
          uriTemplate: https://jira.example.com/browse/$1
receivers:
  keep: []
  strip: []
silenceForm:
  strip:
    labels:
      - job
  defaultAlertmanagers:
    - local
ui:
  refresh: 30s
  hideFiltersWhenIdle: true
  colorTitlebar: false
  minimalGroupWidth: 420
  alertsPerGroup: 5
  collapseGroups: collapsedOnMobile
```