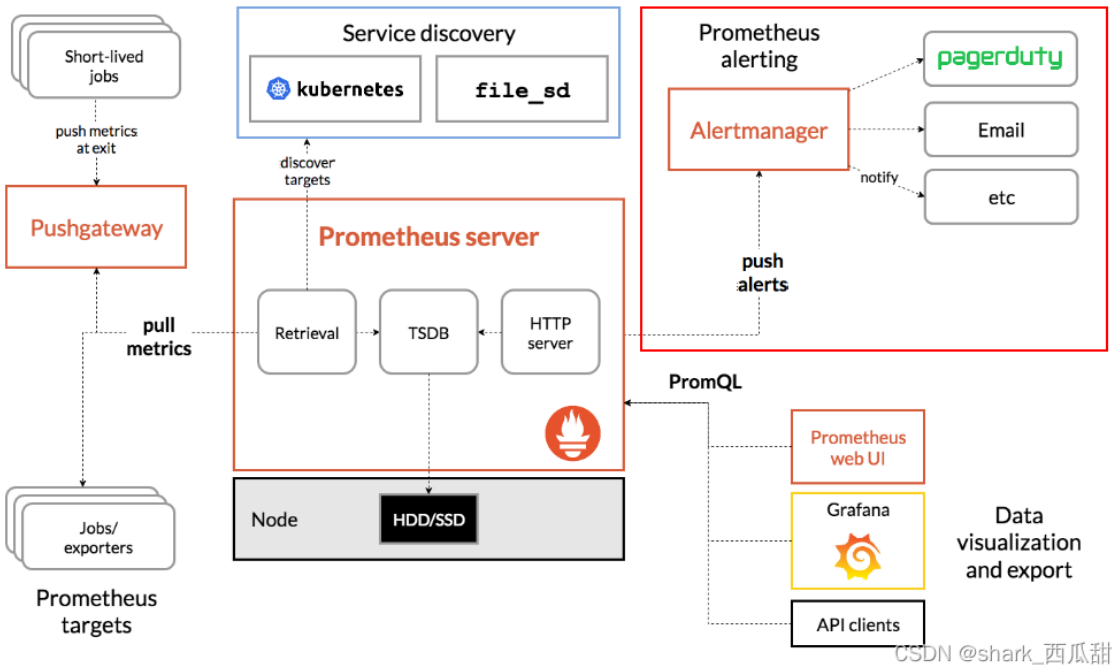
1. 介绍

Alertmanager 用于接收 Prometheus 推送的警报信息，并对这些警报信息进行分组、路由、删除重复警报数据等处理，还可以进一步设置某些警报的静默，抑制等。最后还可以使用 邮件，Webhook 方式等发送通知。

下图中右上角位置是 Alermanager 在整个 Prometheus 架构中的位置。



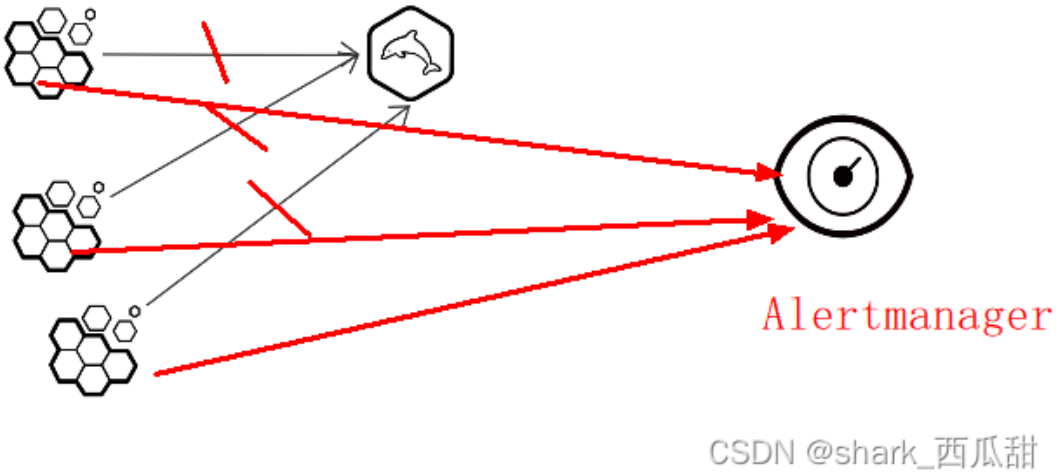
2. 核心概念

2.1. Grouping 分组

分组将性质相似的警报分类到单个通知中。这在多个系统同时发生故障的较大中断期间特别有用，并且数百到数千个警报可能会同时触发。

试想一个部署有 MySQL 的服务器网络中断，导致 k8s 集群中所有应用示例无法连接到这台MySQL。假设 k8s 集群中有成百上千个应用实例，将会同时发送警报信息到 Alertmanager。

其实作为管理者，只想获得一个警报信息即可，并且可以看到都有哪些实例受到影响。因此 Alertmanager 可以配置按机器和警报名称进行分组，以便发送一条比较紧凑、有汇总的通知。



2.2. Inhibition 抑制

抑制是指如果一些其他警报已经启动，则抑制另外一些警报的通知的概念。一些其他警报和另外一些警报是存在一定关联性的。

例如：正在触发警报，通知无法访问整个群集。Alertmanager可以配置为，如果该特定警报正在启动，则将与该集群有关的所有其他警报静音。这将阻止数百或数千个与实际问题无关的触发警报的通知。

抑制可通过Alertmanager的配置文件进行配置。

2.3. Silences 静默

静默是在给定时间内简单地静默警报的一种简单方法。
使用场景，当升级某些服务时候，但是有不想修改关于这些服务的高级规则的配置，可以将这些服务的告警通知设置为在升级的时间段内处于静默状态，静默状态是有告警被触发，但是不发送告警通知到媒介。

静默是基于匹配器配置的，就像路由树一样。将检查传入警报是否与激活状态的静默的所有条件相等或正则表达式相匹配。如果符合或者匹配，则不会发出该警报的通知。

静默可以在配置文件中配置，也可以在Alertmanager的web界面中配置的。

3. 部署

3.1 二进制方式

3.1.1 下载

在 Prometheus 下载页面可以找到不同的版本和适合不同平台的二级制部署包
这里以 Linux amd64 为例。

```
curl -o alertmanager.tgz -L
https://github.com/prometheus/alertmanager/releases/download/v0.27.0/alertmanage
r-0.27.0.linux-amd64.tar.gz
```

3.1.2 部署

```
tar -xf alertmanager.tgz -C /usr/local/
ln -s /usr/local/alertmanager-0.27.0.linux-amd64/  /usr/local/alertmanager
```

3.1.3 配置 systemd

命令行启动参数
参数	含义
`--config.file="alertmanager.yml"`	指定配置文件
`--storage.path="data/"`	指定数据存储基础路径
`--data.retention=120h`	数据保留时长
`--web.listen-address=:9093`	用于公开度量和web界面的地址,参数可以使用多次指定多个。
`--web.external-url=WEB.EXTERNAL-URL`	指定一个 FQDN，比如 --web.external-url=http://grafana.host.com ，这样警报信息中会包含查看当时这个警报的图形链接。

命令行的所有配置参数可以使用此命令获取： `alertmanager -h`
/etc/systemd/system/alertmanager.service

```
[Unit]
Description=Alertmanager 警报管理器
After=network-online.target
```

```
Wants=network-online.target

[Service]
#EnvironmentFile=-
ExecStart=/usr/local/alertmanager/alertmanager --
config.file=/usr/local/alertmanager/alertmanager.yml

KillSignal=SIGQUIT

Restart=always

RestartPreventExitStatus=1 6 SIGABRT

TimeoutStopSec=5
KillMode=process
PrivateTmp=true
LimitNOFILE=1048576
LimitNPROC=1048576

[Install]
WantedBy=multi-user.target
```

3.1.4 启动服务

```
systemctl daemon-reload
systemctl enable --now alertmanager.service
```

3.2. docker-compose 方式

```
version: "3.9"
services:
  alertmanager:
    user: "65534"
    image: quay.io/prometheus/alertmanager
    command: --config.file=/etc/alertmanager/alertmanager.yml --
storage.path=/alertmanager --web.listen-address=:9093 --web.listen-address=:9098
    volumes:
      - "./config/:/etc/alertmanager"
      - "./data:/alertmanager"
    network_mode: "host"
    expose:
      # web 端口
      - "9093"
      # 集群端口
      - "9094"
      # 还是 web 端口
      - "9098"
```

4. 配置

配置文件中一般配置静默规则、通知路由，通知接收器等。

Alertmanager 和 Prometheus 一样支持热加载配置，只需要向进程发送信号：`SIGHUP` 或者向服务发送POST 请求：

`curl -X POST http://ip:9093/-/reload`

如果配置文件中有错误，配置文件将不会更新。

4.1. 配置文件介绍

下面是一些配置示例，覆盖了最主要的基础配置, 这里官方文档有完整的配置说明

4.1.1. 全局配置

```
global:
  # 用于发送电子邮件的默认 SMTP 主机 ，包括端口号，注意不是收件人的邮箱地址。
  # 端口号通常为 25 或者 TLS 的 SMTP 为 465
  smtp_smarthost: 'stmpt.126.com:25'
  # 默认的SMTP发件人标头字段
  smtp_from: 'xxx@126.com'

  # 要显示给SMTP服务器的默认主机名。
  [ smtp_hello: '126.com' | default = "localhost" ]

  # 使用CRAM-MD5、LOGIN和PLAIN的SMTP身份验证用户名，一般是邮箱的用户名。
  # 如果为空，则Alertmanager不会对SMTP服务器进行身份验证。如果 SMTP 需要认证，则会失败。
  # 此邮箱需在其邮箱服务器上开通 SMTP 功能。
  [ smtp_auth_username: 'xxx@126.com' ]

  # 身份验证秘钥，不是登录密码。
  [ smtp_auth_password: <secret> ]

  # 发送邮件是否使用 TLS，如果是true ，smtp_smarthost 的值中的端口应该是 465
  # 请注意，Go 不支持到远程 SMTP 服务器的未加密连接。
  [ smtp_require_tls: <bool> | default = true ]

  # 如果警报不包括EndsAt，ResolveTimeout 是alertmanager使用的默认值。
  # 经过这段时间后，如果警报尚未更新，它可以将其声明为已解决。
  # 这对普罗米修斯的警报没有影响，因为它们总是包括 EndsAt。
  [ resolve_timeout: <duration> | default = 5m ]


# 从中读取自定义通知模板定义的文件。
# 最后一个元素可以使用通配符匹配器，例如 'templates/*.tmpl'
templates:
  [ - <filepath> ... ]

# 路由的根节点，称为根路由，所有的警报必须先经过根路由，之后再分配给对应的子路由。
route: <route>

# 通知接收者的列表。这里可以配置邮件、webhook 等接收告警消息者。
receivers:
  - <receiver> ...

# 一个包含了一个或多个抑制规则的列表.
inhibit_rules:
  [ - <inhibit_rule> ... ]
```

```
# 指定可在路由树 "active_time_intervals" 字段中引用的时间间隔名称，以便在一天中的特定时间静音/激活特定路由。
# 具体如何设置，继续参看下面的章节
time_intervals:
  [ - <time_interval> ... ]
```

4.1.2. `receiver` 接收器

4.1.2.1. 接收器基础设置

receiver 是配置一个或多个接收警报消息的地方。

基础配置包含如下字段：

```
receivers:
    # 接收器的唯一名称。
  - name: <string>

    # 配置不同的接收类型，如下仅展示和介绍经常使用的几种。
    email_configs: # 发邮件在这里配置
      [ - <email_config>, ... ]
    webhook_configs: # 利用 webhook 发送钉钉消息等，在这里配置
      [ - <webhook_config>, ... ]
    wechat_configs: # 发送企业微信消息，在这里配置
      [ - <wechat_config>, ... ]
```

4.1.2.2. 接收器集成设置-发送邮件

这些设置允许配置一个到多个特定的接收器。

`<email_config>`

```
# 是否通知已解决的警报。
[ send_resolved: <boolean> | default = false ]

# 接收警报信息的邮件地址，多个用英文逗号分开。例如： 'xxx@qq.com,yyy.@qq.com'
to: <tmpl_string>

# 向接收人展示的发件人的地址。
[ from: <tmpl_string> | default = global.smtp_from ]

# 发送电子邮件的SMTP主机。
[ smarthost: <string> | default = global.smtp_smarthost ]

# 要标识给SMTP服务器的主机名。
[ hello: <string> | default = global.smtp_hello ]

# SMTP身份验证信息
[ auth_username: <string> | default = global.smtp_auth_username ]
[ auth_password: <secret> | default = global.smtp_auth_password ]

# SMTP TLS要求。
# 请注意，Go不支持到远程SMTP终结点的未加密连接
[ require_tls: <bool> | default = global.smtp_require_tls ]

# TLS configuration.
tls_config:
```

```
  [ <tls_config> ]

# 电子邮件通知的HTML正文，这里一般使用定义好的模板，这里的 email.default.html 是模板名称.
# 模板名称是在全局的 "templates" 字段指定的模板文件中定义的。
[ html: <tmpl_string> | default = '{{ template "email.default.html" . }}' ]
# 电子邮件通知的文本正文。
[ text: <tmpl_string> ]

# 其他邮件头电子邮件头键/值对。
[ headers: { <string>: <tmpl_string>, ... } ]
```

示例

```
receivers:
- name: 'team-X-mails'
  email_configs:
  - to: 'team-X+alerts@example.org, team-Y+alerts@example.org'

- name: 'team-X-pager'
  email_configs:
  - to: 'team-X+alerts-critical@example.org'
  pagerduty_configs:
  - routing_key: <team-X-key>

- name: 'team-Y-mails'
  email_configs:
  - to: 'team-Y+alerts@example.org'
    headers:
      subject: "邮件标题"
      from: "邮件来自哪儿"
      to: "发给谁的"
```

4.1.2.3. 接收器集成设置-使用 web hook 发送钉钉

`<webhook_config>`
webhook接收器允许配置通用接收器。

如果发送钉钉，需要配合一个插件 prometheus-webhook-dingtalk

prometheus-webhook-dingtalk 会启动一个服务，并暴露一个接口。

alertmanager 可以通过 `webhook_config` 中的 `url` 字段的配置，向 prometheus-webhook-dingtalk 发送一个 POST 的 HTTP 请求。从而实现将警报信息向钉钉发送的目的。

```
# 是否发送已恢复的警报信息。
[ send_resolved: <boolean> | default = true ]

# 要向其发送HTTP POST请求的 URL。
url: <secret>

# 要包含在单个webhook消息中的最大警报数。超过此阈值的警报将被截断。
# 将其保留为默认值0时，将包括所有警报。
[ max_alerts: <int> | default = 0 ]
```

Alertmanager将以以下JSON格式向配置的端点发送HTTP POST请求：

```
{
  "version": "4",
  "groupKey": <string>,              // 标识警报组的键
  "truncatedAlerts": <int>,          // 由于"max_alerts" 而截断了多少个警报
  "status": "<resolved|firing>",
  "receiver": <string>,
  "groupLabels": <object>,
  "commonLabels": <object>,
  "commonAnnotations": <object>,
  "externalURL": <string>,           // 点击这个链接，可以访问到 Alertmanager
  "alerts": [
    {
      "status": "<resolved|firing>",
      "labels": <object>,
      "annotations": <object>,
      "startsAt": "<rfc3339>",
      "endsAt": "<rfc3339>",
      "generatorURL": <string>,
      "fingerprint": <string>
    },
    ...
  ]
}
```

其实，`webhoook_config` 的配置不仅仅支持钉钉，它是一个通用的 webhook 工具。 有一个与此功能集成的列表。

要实现钉钉告警，需要安装一个钉钉插件的 webhook 服务，具体点击链接
示例:

```
receivers:
  - name: 'web.hook'
    webhook_configs:
      - url: 'http://192.168.3.100:8060/dingtalk/webhook_mention_all/send'
```

4.1.2.4. 接收器集成设置-发送企业微信

`<wechat_config>`
微信通知通过微信API发送。

```
# 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" . }}' ]
```

4.1.3. time_interval 时间间隔

4.1.3.1 `time_interval`

指定可在路由树中引用的命名时间间隔，以表示在一天中的特定时间处于 静音或者激活 状态的特定路由。

```
time_intervals:
  - name: <string>  # 此时间间隔的名称,可以在子路由中被引用。
    time_intervals:
      # 下面定义一个一个的时间间隔
      [ - <time_interval_spec> ... ]
```

4.1.3.2 `time_interval_spec`

包含时间间隔的实际定义。语法支持以下字段：

```
time_intervals:
  - name: <string>  # 此时间间隔的名称,可以在子路由中被引用。
    time_intervals:
      - times:
        [ - <time_range> ...]            // 小时分钟
        weekdays:
        [ - <weekday_range> ...]         // 周
        days_of_month:
        [ - <days_of_month_range> ...]   // 天
        months:
        [ - <month_range> ...]   // 月
        years:
        [ - <year_range> ...]    // 年
        location: <string>  // 时区
```

所有字段都是列表。在每个非空列表中，必须满足至少一个元素才能与字段匹配。

如果未指定字段，则任何值都将与该字段匹配,比如未指定 `days_of_month` 字段，那么已经配置的 `times` 字段的值将会匹配为当月的每一天 中的时间段。

为了使某个瞬间的时间与完整的时间间隔相匹配，所有字段都必须匹配。

如果未指定时区，则时间以UTC为单位。

一些字段支持区间和负指数，详情如下。

4.1.3.3 `time_range`

包含开始时间，但不包含结束时间的范围，可以方便地表示在小时边界上开始/结束的时间。
例如，`start_time:'20:00'` 和 `end_time:'24:00'` 将从 20:00 开始，并在 24:00 之前结束。
它们配置格式是这样的：

```
time_intervals:
  - name: <string>  # 此时间间隔的名称,可以在子路由中被引用。
    time_intervals:
      - times:
        - start_time: HH:MM
          end_time: HH:MM
```

示例：

```
time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - times:
        - start_time: 20:00
          end_time: 24:00
```

如上示例中没有其他时间字段，时间范围就表示每天的 20:00 到 23:59:59之间

4.1.3.4 `weekday_range`

一周中的日期列表，一周从周日开始，到周六结束。应按名称指定日期（例如“Sunday”）。

中文	英文（英式发音）	缩写	手表缩写
星期一	Monday ['mʌndeɪ]	Mon.	MON/MO
星期二	Tuesday ['tju:zdeɪ]	Tue.	TUE/TU
星期三	Wednesday ['wenzdeɪ]	Wed.	WED/WE
星期四	Thursday ['θɜ:zdeɪ]	Thur.	THU/TH
星期五	Friday ['fraideɪ]	Fri.	FRI/FR
星期六	Saturday ['sætədeɪ]	Sat.	SAT/SA
星期天	Sunday ['sʌndeɪ]	Sun.	SUN/SU

为了方便起见，还接受形式为 `<start_day>:<end_day>` 的范围，并且范围两端都包含。
例如：要表示: 星期一到星期三，星期六，星期日，应写成：`['monday:wednesday','saturday', 'sunday']`

```
time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - weekday_range:
        - 'monday:wednesday'
        - 'saturday'
        - 'sunday'
        times:
        - start_time: 20:00
          end_time: 24:00
```

如上示例只有 `times` 和 `weekday_range` 字段，时间范围表示：

- 每周的周一到周三的 20：00 到23:59:59之间
- 还有每周六和每周日的 20：00 到23:59:59之间。

4.1.3.5 `days_of_month_range`

一个月中的数字天数列表。

天数从1开始。

负值也可以接受，从月底开始，例如1月份的-1表示1月31日。

例如：`['1:5', '-3:-1']`。超过月初或月底将导致它被抑制。

例如，在2月份指定 `['1:31']` 将根据闰年将实际结束日期固定为28或29。包括两端。

```
time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - days_of_month_range:
        - '1:31'
        times:
        - start_time: 20:00
          end_time: 24:00
```

如上示例表示：每个月的月初到月底的每一天的 20：00 到23:59:59之间。

4.1.3.6 `month_range`

由不区分大小写的名称（例如“January”）或数字标识的日历月份列表，其中January=1。

一月到十二月的英文

一个年(Year)有十二个月份(Month),从一月到十二月的英文、发音及缩写如下表：

中文	英文（英式发音）	缩写
一月	January ['dʒænjʊərɪ]	Jan.
二月	February [febrʊərɪ]	Feb.
三月	March [mɑtʃ]	Mar.
四月	April ['eɪprəl]	Ari.
五月	May [meɪ]	May.
六月	June [dʒu:n]	Jun.
七月	July [dʒʊ'laɪ]	Jul.
八月	August [ɔ'gəst]	Aut.
九月	September [sep'tembə]	Sep.
十月	October [ɒk'təʊbə]	Oct.
十一月	November [nəʊ'vembə]	Nov.
十二月	December [dɪ'sembə]	Dec.

也可接受范围。例如，`['1:3', 'may:august', 'december']`。包括两端。

```yaml
time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - month_range:
        - '1:3'
        - 'may:august'
        - 'december'
        days_of_month_range:
        - '1:31'
        times:
        - start_time: 20:00
          end_time: 24:00
```

以上示例表示：每年的 1月份到3月份和5月份到8月份以及12月份的每一天的 20：00 到23:59:59之间。

4.1.3.7 `year_range`

年份的数字列表。接受范围。例如，`["2020:2022", "2030"]`。两端均包含。
不指定此字段，表示匹配到每一年。通常不指定。

4.1.3.8 `location`

与IANA时区数据库中的位置匹配的字符串，例如 东八区，即中国时区：`Asia/Shanghai`。

```yaml
time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - location: 'Asia/Shanghai'
        month_range:
        - '1:3'
        - 'may:august'
        - 'december'
        days_of_month_range:
        - '1:31'
        times:
        - start_time: 20:00
          end_time: 24:00
```

您也可以使用 `'Local'` 表示使用 **Alertmanager** 运行的服务器作为本地时间的位置, 或使用 `'UTC'` 表示 **UTC** 时间。

如果未提供时区，则时间间隔以 **UTC** 时间为单位。

注意：在Windows上，只有 `'Local '` 或 `'UTC'` 才受支持，除非您使用 `ZONEINFO` 环境变量提供自定义时区数据库。

关于中国时区，可以参考这篇知乎的文章 https://zhuanlan.zhihu.com/p/450867597

4.1.4. 路由相关配置

通过与路由相关的设置，可以根据时间配置警报的路由、聚合、抑制和静音方式。

4.1.4.1. `<route>` 根路由

`route` 块定义路由树中的节点及其子节点。

如果子节点中的某些可选配置未设置，那些可选的配置参数（就是用中括号括起来的配置参数）将从其父节点继承。

每个警报都进入配置的顶级路由的路由树，该路由树必须匹配所有警报（即无需配置任何匹配器）。

然后它遍历子节点。如果子节点中的 `continue` 设置为 `false`，则在第一个匹配成功的子路由之后就停止继续向下匹配。

如果子节点中的 `continue` 为 `true`，则警报将继续与后续同级路由匹配。

如果警报与节点的任何子节点都不匹配，则会根据当前节点的配置参数来处理警报,一般会由根 `route` 块中配置的 `receiver` 处理。

```
# 根路由，每个传入的警报都会经过这里
route:
  # 根路由不能有任何匹配器，因为它是所有警报的入口点。
  # 它需要配置一个默认的接收器，以便将与任何子路由不匹配的警报发送给此接收器。
  # 其值需要是全局配置块: receivers 中配置的其中一个。
  [ receiver: <string> ]

  # 传入的警报，依据这里定义的标签进行分组，标签值相同的被分为一组，组名就是标签名。
  # 例如 含有 cluster=A 和 alertname=LatencyHigh 标签的多个警报将被批处理到一个组中。
  group_by: ['alertname', 'cluster']
  # 若要按所有可能的标签进行聚合，请使用特殊值"..."作为唯一的标签名称，例如：group_by:
['...']
  # 这有效地完全禁用了聚合，按原样传递所有警报。
  # 这不太可能是你想要的，除非你的警报量很低，或者你的上游通知系统执行自己的分组。



  # 当一个组中的第一条警报，至少等待 "group_wait" 时间后再发送初始通知。
  # 这种方式可以确保您在 "group_wait" 时间内获得同一组的多个警报，对这条新的告警做一些分组、
更新、静默的操作之后，再一起开始触发。
  group_wait: 10s

  # 当一个组的初始警报已经发送，如果这个组中再来新的警报，就等待 group_interval 时间后再发送
警报通知，并且 repeat_interval 的起始时间被刷新为当前时间，并重新计算。
  # 通常 5m 或者更长
  group_interval: 5m



  # 请注意，此参数由Alertmanager的“--data.reduration”配置标志隐式绑定。
  # 重复警报发送的时间间隔，就是如果一组警报，一直没有恢复，并且组内没有新的警报，将在这个时间之
后重新发送通知，也就是再次发送通知的间隔时间，
  # repeat_interval 不应小于 group_interval， 一组警报中有新的警报加入，则这个时间会重新
计算。
  repeat_interval:  3h

  # 警报是否应继续匹配后续同级路由节点。
  [ continue: <boolean> | default = false ]

  # 警报必须满足的匹配器列表才能匹配节点，一般在子路由中设置。
  matchers:
    [ - <matcher> ... ]

  # 设置路由应该静默的时间。
  # 与全局设置 "time_intervals" 字段中定义的静默时间间隔的名称相匹配。
  # 此外，根节点不能有任何静默时间。
```

```
  # 当路由被静默时，它不会发送任何通知，但在其他情况下会正常工作（包括如果未设置“continue”选项，则结束路由匹配过程）
  mute_time_intervals:
    [ - <string> ...]

  # 路由应处于活动状态的时间。它们必须与全局设置 "time_intervals" 字段中定义的时间间隔的名称相匹配。
  # 空值表示路线始终处于活动状态。 此外，根节点不能有任何 active times。
  # 路由将仅在活动时发送通知，但在其他情况下正常运行（包括如果未设置 "continue" 选项，则结束路由匹配过程）。
  active_time_intervals:
    [ - <string> ...]

  # 以上所有属性都由所有子路由继承，并且可以在每个路由上覆盖。
  # 一个或者更多子路由
  routers:
    [ - <router> ...]
```

简单示例：

```yaml
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname']
  group_wait: 2s
  group_interval: 2s
  repeat_interval: 5s
  receiver: 'web.hook'
  time_intervals:
  - name: uptimeforapp01
    time_intervals:
      - location: 'Asia/Shanghai'
        month_range:
        - '1:3'
        - 'may:august'
        - 'december'
        days_of_month_range:
        - '1:31'
        times:
        - start_time: 20:00
          end_time: 24:00
  routes:
    - matchers:
        - job = java-top
#      receiver: web.hook
#      continue: true
#    - matchers:
#        - project = smyy
#      receiver: web.hook.1
#      group_by: ['instance']
#      continue: true
 #   - matchers:
 #       - service = java
 #     receiver: web.hook.2
receivers:
  - name: 'web.hook'
```

```
    webhook_configs:
      - url: 'http://127.0.0.1:8070/dingtalk/webhook_mention_all/send'
```

4.1.4.2. 标签匹配器 `matcher`

标签匹配器是用在 routes and inhibition rules 之间进行警报的匹配。
也就是使用标签匹配器配置了匹配的规则，如果哪个警报中所包含的标签和配置的规则相符合，就代表此警报是要处理的警报，如果某个 子路由中使用此 匹配器，那就是此子路由要处理这个警报，如果某个 inhibition rules 中使用了此匹配器，那就是这个 inhibition rules 处理此警报。

匹配器是一个字符串，具体语法由三个标记组成：

- 有效的标签名
- `=, !=, =~, or !~` 中其中的一个运算符。`=~` 和 `!~` 用于正则表达式的 匹配和不匹配。
- 运算符后面是 UTF-8 字符串，可以用双引号括起来。在每个标记之前或之后，可能存在任意数量的空白。

运算符右边的内容可以是空字符串，支持反斜杠转义规则。

当有多个匹配器规则时候，他们的书写语法是将每一个规则放在一个 yaml 语法的列表中，它们之间的关系是 and。这意味着在针对给定警报上的标签进行测试时，匹配器的所有规则都必须匹配。例如，具有以下标签的警报：

```
{"project":"beijing","service":"mysql"}
```

必须使用如下匹配器规则进行匹配

```
route:
  routers:
    - matchers:
        - project = beijing
        - service = mysql
```

下面使用正则表达式的匹配规则也是可以的

```
route:
  routers:
    - matchers:
        - project = beijing
        - service =~ "mysql|postgresql"
```

当然将上述匹配规则写成一行，使用 `[]` 进行包裹也是可以的。

```
route:
  routers:
    - matchers:[project = beijing, service =~ "mysql|postgresql"]
```

将匹配规则内层在包上一层大括号也是支持的。

```
route:
  routers:
    - matchers:['{project=beijing, service=~ "mysql|postgresql"}']
```

4.1.4.3. `routers`子路由配置

```
  # 具体的一个一个的子路由树.
  routes:
  # 此子路由使用正则表达式进行匹配，凡是警报的标签 service 的值是 foo1 或者 foo2 或者 baz 的警报就会
  # 走这个子路由
  - matchers:
    - service = ^(foo1|foo2|baz)$

    # 凡是走这个子路由的警报，使用如下的接收器接收警报信息， team-X-mails 是在全局中的 receivers 配置块中定义的
    # 一个接收器名称。
    receiver: team-X-mails

    # 子路由中可以嵌套一个或者多个子路由
    # 当这个子路由中出现包含并且匹配标签 "severity" 的值是 "critical" 的，
    # 将警报信息从 "team-X-pager" 接收器发送出去。
    # 否则还是返回给父级路由中定义的接收器 "team-X-mails" 发送警报。
    routes:
    - matchers:
        - severity = critical
      receiver: team-X-pager

  - matchers:
      - service = files
    receiver: team-Y-mails

    # 同样这个是在符合 "service" 的值是 "files" 中的警报中，再挑选出配置 "severity: critical" 的警报
    # 从接收器 team-Y-pager 发送警报信息。否则继续返回给父级定义的接收器 team-Y-mails 发送出去警报信息。
    routes:
    - matchers:
        - severity = critical
      receiver: team-Y-pager

  # This route handles all alerts coming from a database service. If there's
  # no team to handle it, it defaults to the DB team.
  - matchers:
      - service = database

    receiver: team-DB-pager
    # Also group alerts by affected database.
    group_by: [alertname, cluster, database]

    routes:
    - matchers:
        - owner = team-X
      receiver: team-X-pager

    - matchers:
        - owner = team-Y
      receiver: team-Y-pager
```

示例：

```yaml
#根路具有的所有参数，如果子路由中未设置这些参数，则这些参数将由子路由继承。
route:
  group_wait: 5s
  group_interval: 1m
  repeat_interval: 30m
  group_by: [cluster, alertname]
  time_intervals:
  - name: offhours
    time_intervals:
     - location: 'Asia/Shanghai'
       weekday_range:
       - 'monday:friday'
       times:
       - start_time: 18:00
         end_time: 08:00


  - name: holidays
    time_intervals:
      - location: 'Asia/Shanghai'
        weekday_range:
        - 'saturday:sunday'
  receiver: 'default-receiver'
  # 所有与以下子路由不匹配的警报都将保留在根节点上，并使用接收器 "default-receiver" 发送警报信息。
  routes:
  # 所有service=mysql或service=casandra的警报都使用接收器 "database-pager" 发送警报信息
  - receiver: 'database-pager'
    group_wait: 10s
    matchers:
    - service=~"mysql|cassandra"
  #所有带有 team="frontend" 标签的警报都匹配此子路由.
  # 它们按 product 和 environment 分组，而不是按根路由中定义的 cluster 和 alertname 分组。
  # 并使用接收器 "frontend-pager" 发送警报信息。
  - receiver: 'frontend-pager'
    group_by: [product, environment]
    matchers:
    - team="frontend"

  # 所有带有service=inhouse服务标签的警报都与此子路由匹配。
  # 该路线将在下班时间和节假日期间静音。
  # 即使匹配，它也将继续到下一个子路线
  - receiver: 'dev-pager'
    matchers:
      - service="inhouse-service"
    mute_time_intervals:
      - offhours
      - holidays
    continue: true

    # 所有带有  service=inhouse-service 标签的警报都与此子路由匹配
    # 该路由将仅在非工作时间和节假日期间有效。
  - receiver: 'on-call-pager'
    matchers:
```

```
    - service="inhouse-service"
  active_time_intervals:
    - offhours
    - holidays
```

4.1.5. 抑制相关配置 Inhibition

这允许在系统或服务之间建立依赖关系，以便在中断期间，仅发送一组互连警报中最相关的警报。例如，一台服务器网络中断，同时也配置的这台服务器上的相关服务的状态检查等警报，那只需要发送此服务器网络中断的警报即可,服务器上的服务的状态告警可以被处理为 静音，因为服务器上的服务状态的检查依赖此服务器的网络。

`<inhibit_rules>`

抑制规则允许在一个或多个匹配了 source_matchers 定义的匹配规则的警报正在触发的情况下，使一组匹配了 target_matchers 定义的匹配规则的警报静默。

从语义上讲，缺失的标签和值为空的标签是一回事。因此，如果源警报和目标警报中都缺少以等号列出的所有标签名称，则将应用禁止规则,因为这会将所有的警报静音。

为了防止警报抑制自身，与规则的目标端和源端匹配的警报不能被相同情况的警报（包括自身）抑制。但是，我们建议选择目标和源匹配器，以使警报永远不会同时匹配双方。它更容易推理，而且不会引发这种特殊情况。

```
# 使抑制生效的匹配规则。
# 至少需要有一个警报完全匹配如下列表中的匹配规则，抑制才会生效。
source_matchers:
  [ - <matcher> ... ]
# 同时满足的如下列表中匹配规则的所有警报将会被静音。
target_matchers:
  [ - <matcher> ... ]

# 匹配 source_matchers 的源警报和匹配 target_matchers 的目标警报中，它们的标签名必须都含有 equal 值中的标签名,且标签值必须相等，才会使抑制生效。
[ equal: '[' <labelname>, ... ']' ]
```

示例

```
#如果同一警报的级别已经是严重级别，我们将任何 warning 级别的通知都设为静默。
inhibit_rules:
- source_matchers:
    - severity="critical"
  target_matchers:
    - severity="warning"
  # 如果警报名称相同，则应用抑制。
  # 注意：
  #    如果源警报和目标警报中都缺少“equal”中列出的所有标签名称，则将应用抑制规则！
  equal: ['alertname']
```

4.2. 检查配置文件

```
amtool check-config alertmanager.yml
```

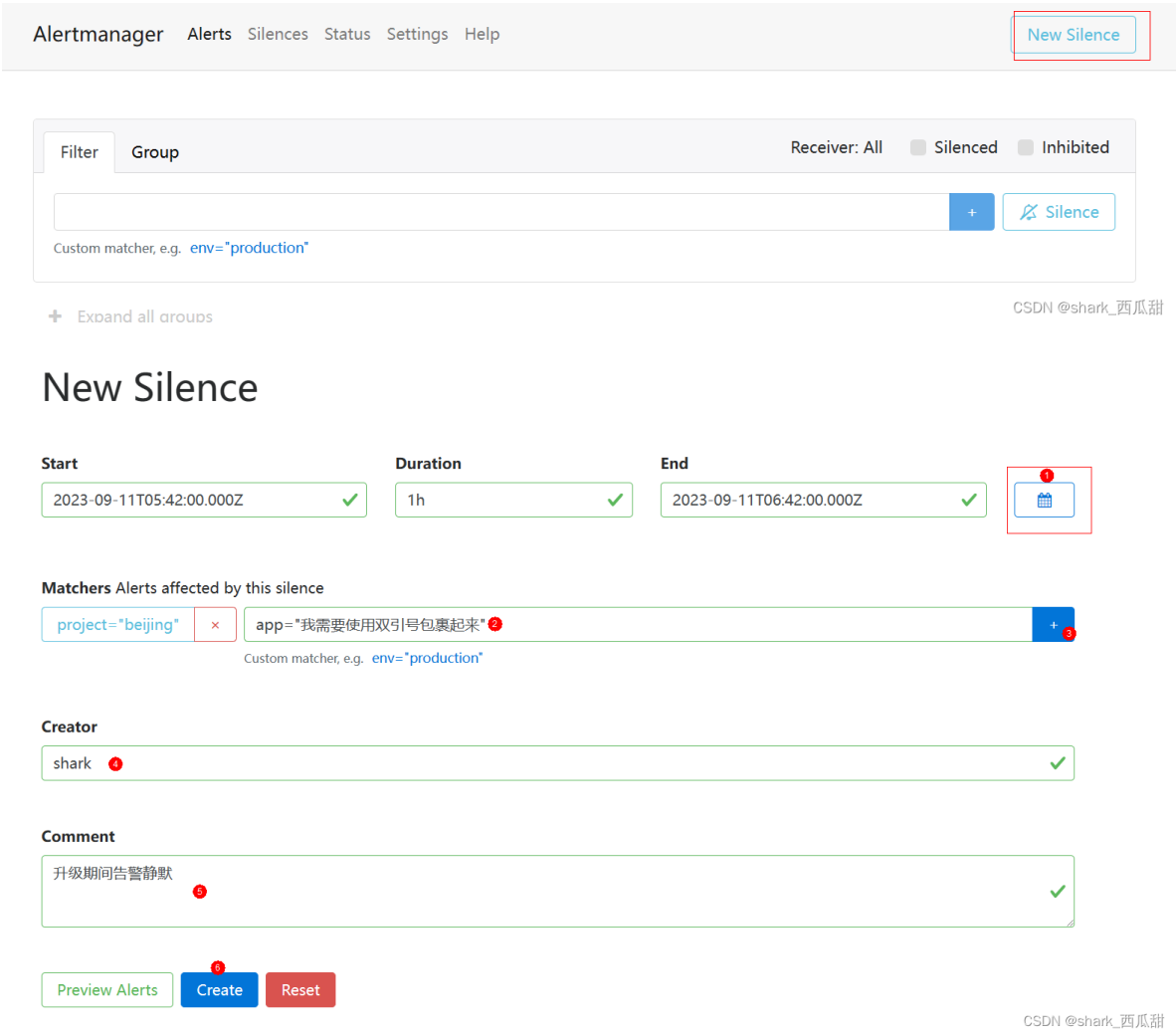
```
[root@db1 alertmanager-0.25.0]# ./amtool check-config alertmanager.yml
Checking 'alertmanager.yml'  SUCCESS
Found:
 - global config
 - route
 - 0 inhibit rules
 - 1 receivers
 - 1 templates
 SUCCESS
```


4.3. 静默配置

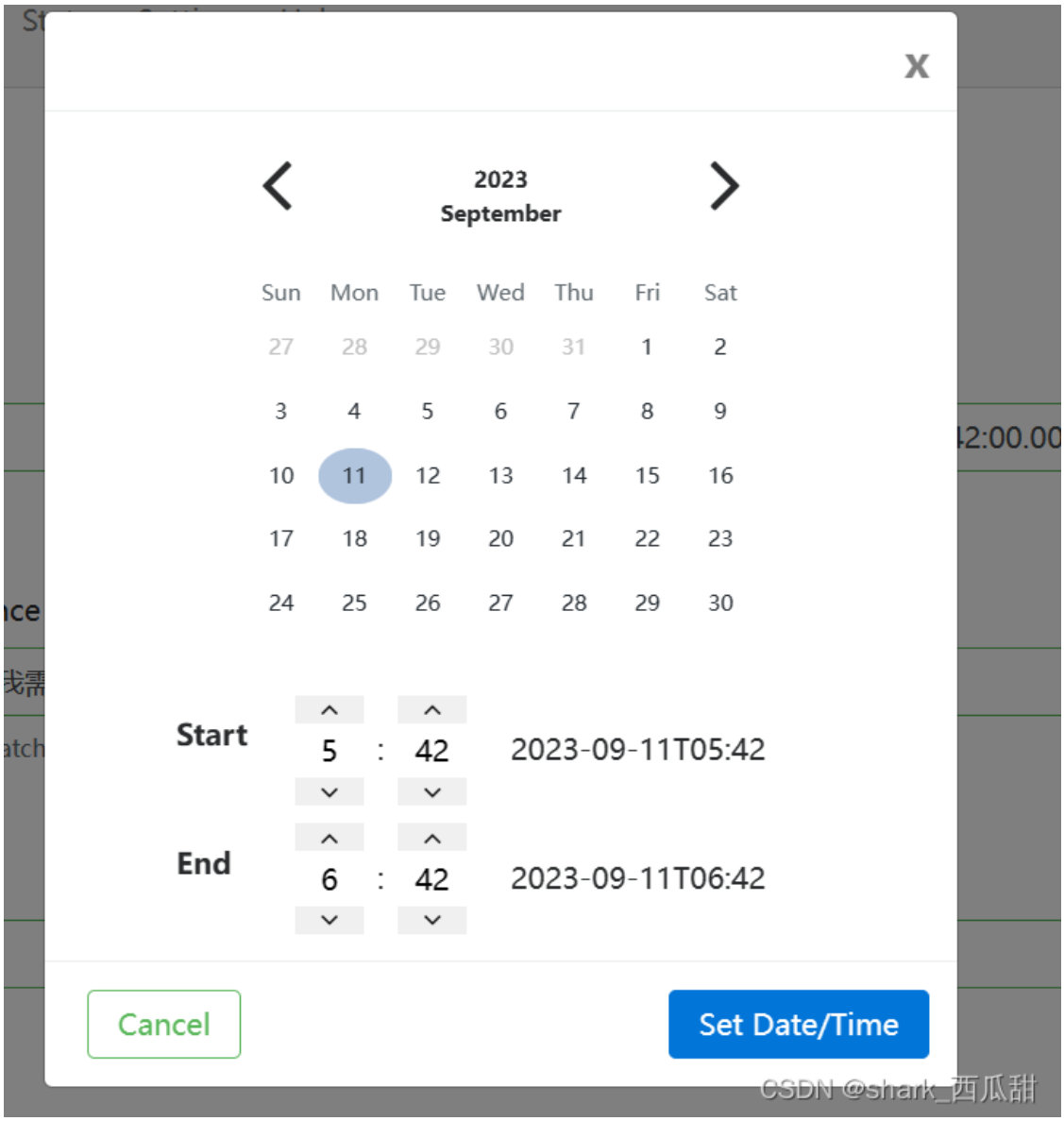
静默是可以让一个或多个警报在指定的某个时间段进入静默状态。

适用于（但不限于）如下场景：

- 升级服务时候，对所要升级的服务设置在升级的时间段不发送警报通知。
- 在服务迁移的时候，对涉及到的服务在迁移的时间段不发送警报通知。



- ① 点击日历图标，选择开始时间和结束时间



- ② Matchers 中填写的是匹配标签的规则，值需要使用双引号引起来
- ③ Creater 填写创建者，可以任意写
- ④ 点击 + 进行添加，可以多次添加
- ⑤ Comment 填写关于这个静默的说明文字
- ⑥ 最后点击 Create

添加之后，再次点击 Silences 会看到所有的 静默列表。