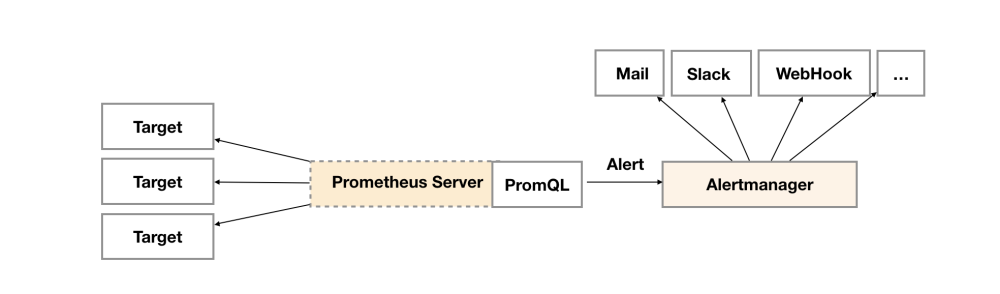
# Prometheus告警处理

# 1.Alertmanager介绍

Prometheus 包含一个报警模块，就是 AlertManager，Alertmanager 主要用于接收 Prometheus 发送的告警信息，它支持丰富的告警通知渠道，而且很容易做到告警信息进行去重，降噪，分组等，是一款前卫的告警通知系统。

通过在 Prometheus 中定义告警规则，Prometheus会周期性的对告警规则进行计算，如果满足告警触发条件就会向Alertmanager 发送告警信息。

告警能力在Prometheus的架构中被划分成两个独立的部分。如下所示，通过在Prometheus中定义AlertRule（告警规则），Prometheus会周期性的对告警规则进行计算，如果满足告警触发条件就会向Alertmanager发送告警信息。



在Prometheus中一条告警规则主要由以下几部分组成：

* 告警名称：用户需要为告警规则命名，当然对于命名而言，需要能够直接表达出该告警的主要内容

* 告警规则：告警规则实际上主要由PromQL进行定义，其实际意义是当表达式（PromQL）查询结果持续多长时间（During）后出发告警

# 2.部署AlertManager

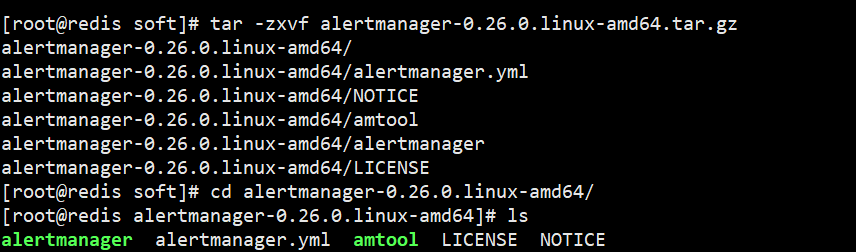
Alertmanager和Prometheus Server一样均采用Golang实现，并且没有第三方依赖。

## Linux安装AlertManager

**1）下载安装包**

Alertmanager最新版本的下载地址可以从Prometheus官方网站<https://prometheus.io/download/>获取。

wget https://github.com/prometheus/alertmanager/releases/download/v0.26.0/alertmanager-0.26.0.linux-amd64.tar.gz



**2）创建alertmanager配置文件**

Alertmanager解压后会包含一个默认的alertmanager.yml配置文件，内容如下所示：

route:  
 group\_by: ['alertname']  
 group\_wait: 30s  
 group\_interval: 5m  
 repeat\_interval: 1h  
 receiver: 'web.hook'  
receivers:  
 - name: 'web.hook'  
 webhook\_configs:  
 - url: 'http://127.0.0.1:5001/'  
inhibit\_rules:  
 - source\_match:  
 severity: 'critical'  
 target\_match:  
 severity: 'warning'  
 equal: ['alertname', 'dev', 'instance']

Alertmanager主要负责对Prometheus产生的告警进行统一处理，因此在Alertmanager配置中一般会包含以下几个主要部分：

* 全局配置（global）：用于定义一些全局的公共参数，如全局的SMTP配置，Slack配置等内容；

* 模板（templates）：用于定义告警通知时的模板，如HTML模板，邮件模板等；

* 告警路由（route）：根据标签匹配，确定当前告警应该如何处理；

* 接收人（receivers）：接收人是一个抽象的概念，它可以是一个邮箱也可以是微信，Slack或者Webhook等，接收人一般配合告警路由使用；

* 抑制规则（inhibit\_rules）：合理设置抑制规则可以减少垃圾告警的产生

**3) 启动Alertmanager**

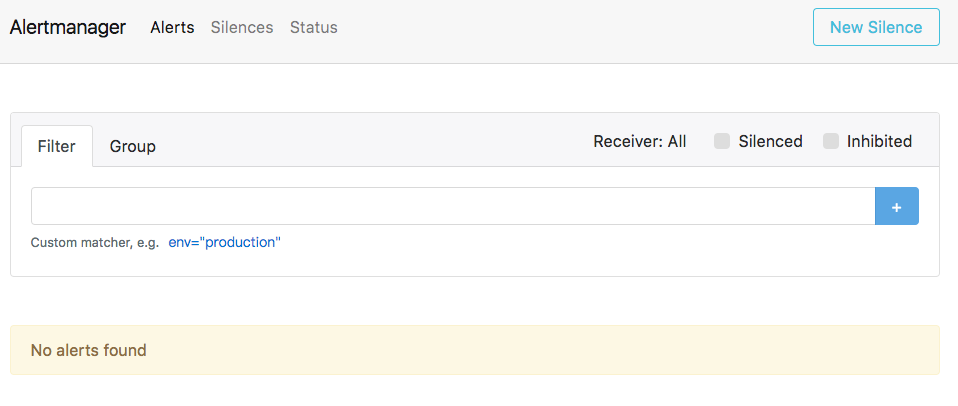
Alermanager会将数据保存到本地中，默认的存储路径为data/。因此，在启动Alertmanager之前需要创建相应的目录：

./alertmanager

用户也在启动Alertmanager时使用参数修改相关配置。--config.file用于指定alertmanager配置文件路径，--storage.path用于指定数据存储路径

**查看运行状态**

Alertmanager启动后可以通过9093端口访问，http://192.168.3.100:9093



Alert菜单下可以查看Alertmanager接收到的告警内容。Silences菜单下则可以通过UI创建静默规则。进入Status菜单，可以看到当前系统的运行状态以及配置信息。

## docker 安装AlertManager

mkdir -p /etc/alertmanager/  
mkdir -p /etc/alertmanager/template  
  
vim /etc/alertmanager/alertmanager.yml  
route:  
 group\_by: ['alertname']  
 group\_wait: 30s  
 group\_interval: 5m  
 repeat\_interval: 1h  
 receiver: 'web.hook'  
receivers:  
 - name: 'web.hook'  
 webhook\_configs:  
 - url: 'http://127.0.0.1:5001/'  
inhibit\_rules:  
 - source\_match:  
 severity: 'critical'  
 target\_match:  
 severity: 'warning'  
 equal: ['alertname', 'dev', 'instance']  
  
docker run -d -p 9093:9093 \  
-v /etc/alertmanager:/etc/alertmanager \  
-v /etc/localtime:/etc/localtime \  
prom/alertmanager

# 3.使用Receiver接收告警信息

告警接收器可以通过以下形式进行配置：

receivers:  
 - <receiver> ...

每一个receiver具有一个全局唯一的名称，并且对应一个或者多个通知方式。

目前官方内置的第三方通知集成包括：邮件、 即时通讯软件（如Slack、Hipchat）、移动应用消息推送(如Pushover)和自动化运维工具（例如：Pagerduty、Opsgenie、Victorops）。Alertmanager的通知方式中还可以支持Webhook，通过这种方式开发者可以实现更多个性化的扩展支持。

## 集成qq邮箱

**qq邮箱申请授权码**

参考：<https://service.mail.qq.com/detail/0/75>



**配置qq邮箱alert**

global: ##全局配置  
 resolve\_timeout: 5m ##当告警的状态由firing变为resolve时，需等待5min，如果报警未更新，则声明该告警已解决。可略微调高阈值避免  
 smtp\_smarthost: 'smtp.qq.com:465'  
 smtp\_from: '276\*\*\*\*211@qq.com'  
 smtp\_auth\_username: '276\*\*\*\*211@qq.com'  
 smtp\_auth\_password: 'xxxxxxxxx' # 这个授权码需要生成，非QQ密码  
 smtp\_require\_tls: false  
templates:  
 - '/etc/alertmanager/template/\*.tmpl'  
route:  
 group\_by: ['service','alertname','cluster'] ##根据label进行分组。--cluster可创建集群  
 group\_wait: 30s ##触发告警后，等待30s发送  
 group\_interval: 10s ##两组告警发送的时间间隔  
 repeat\_interval: 5m ##重复告警发送的时间间隔  
 receiver: 'email' #默认接收者  
receivers: ###告警的处理方式email  
 - name: 'email'  
 email\_configs: ##告警转发到对应邮箱地址  
 - to: '276\*\*\*\*211@qq.com'  
 send\_resolved: true  
 html: '{{ template "email.html" . }}' #使用自定义的模板发送  
   
inhibit\_rules: ###添加抑制规则  
 - source\_match: ####根据label匹配源告警  
 severity: 'critical'  
 target\_match: ####根据label匹配目的告警  
 severity: 'warning'  
 equal: ['alertname', 'dev', 'instance']

**创建自定义模版**

vim /etc/alertmanager/template/email.tmpl  
  
{{ define "email.html" }}  
{{ range $i, $alert :=.Alerts }}  
========监控报警==========<br>  
告警状态：{{ .Status }}<br>  
告警级别：{{ $alert.Labels.severity }}<br>  
告警类型：{{ $alert.Labels.alertname }}<br>  
告警应用：{{ $alert.Annotations.summary }}<br>  
告警主机：{{ $alert.Labels.instance }}<br>  
告警详情：{{ $alert.Annotations.description }}<br>  
触发阀值：{{ $alert.Annotations.value }}<br>  
告警时间：{{ ($alert.StartsAt.Add 28800e9).Format "2006-01-02 15:04:05"}}<br>  
========end=============<br>  
{{ end }}  
{{ end }}

注意：

StartsAt.Fromat为UTC时间，比北京时间晚了8个小时 需要加 28800e9

2006-01-02 15:04:05 不能改变 ，此处为go语言出版时间

# 4.启用alertmanager模块和rules告警规则

在Prometheus的架构中被划分成两个独立的部分。Prometheus负责产生告警，而Alertmanager负责告警产生后的后续处理。因此Alertmanager部署完成后，需要在Prometheus中设置Alertmanager相关的信息。

1) 编辑Prometheus配置文件prometheus.yml,并添加以下内容

global:  
 scrape\_interval: 5s # Set the scrape interval to every 15 seconds. Default is every 1 minute.  
 evaluation\_interval: 5s # Evaluate rules every 15 seconds. The default is every 1 minute.  
 # scrape\_timeout is set to the global default (10s).  
  
# Alertmanager configuration  
alerting:  
 alertmanagers:  
 - static\_configs:  
 - targets:  
 - 192.168.3.100:9093  
  
# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.  
rule\_files:  
 - "/etc/prometheus/rules/\*.rules"  
 # - "second\_rules.yml"  
# A scrape configuration containing exactly one endpoint to scrape:  
# Here it's Prometheus itself.

重启Prometheus服务，成功后，可以从http://192.168.3.100:9090/config查看alerting配置是否生效。

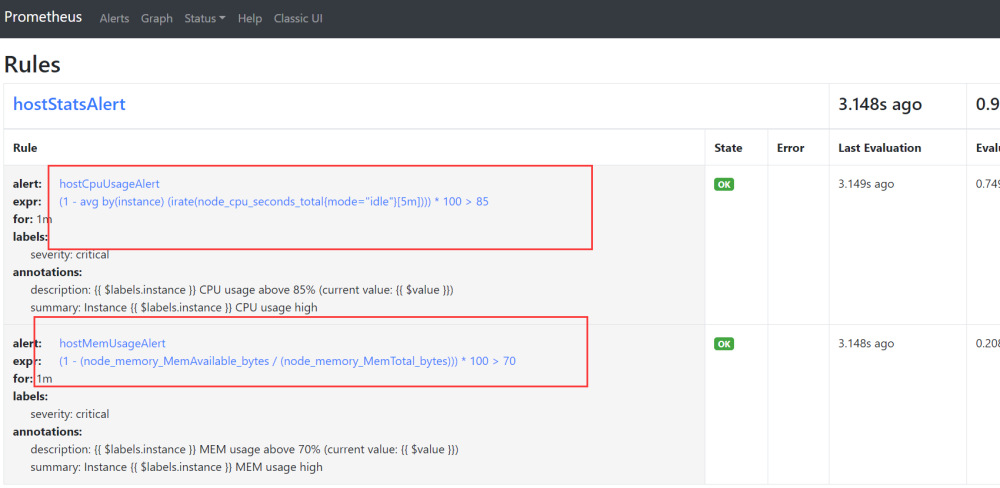
**2) 创建rules告警规则**

主机CPU利用率>85%

主机MEM利用率>70%

mkdir -p /etc/prometheus/rules  
vim /etc/prometheus/rules/alerts.rules  
  
groups:  
- name: hostStatsAlert  
 rules:  
 - alert: hostCpuUsageAlert  
 expr: (1 - avg(irate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance))\*100 > 85  
 for: 1m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Instance {{ $labels.instance }} CPU usage high"  
 description: "{{ $labels.instance }} CPU usage above 85% (current value: {{ $value }})"  
 - alert: hostMemUsageAlert  
 expr: (1 - (node\_memory\_MemAvailable\_bytes{} / (node\_memory\_MemTotal\_bytes{})))\* 100 > 70  
 for: 1m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Instance {{ $labels.instance }} MEM usage high"  
 description: "{{ $labels.instance }} MEM usage above 70% (current value: {{ $value }})"

重启Prometheus后访问Prometheus UI http://127.0.0.1:9090/rules可以查看当前以加载的规则文件。



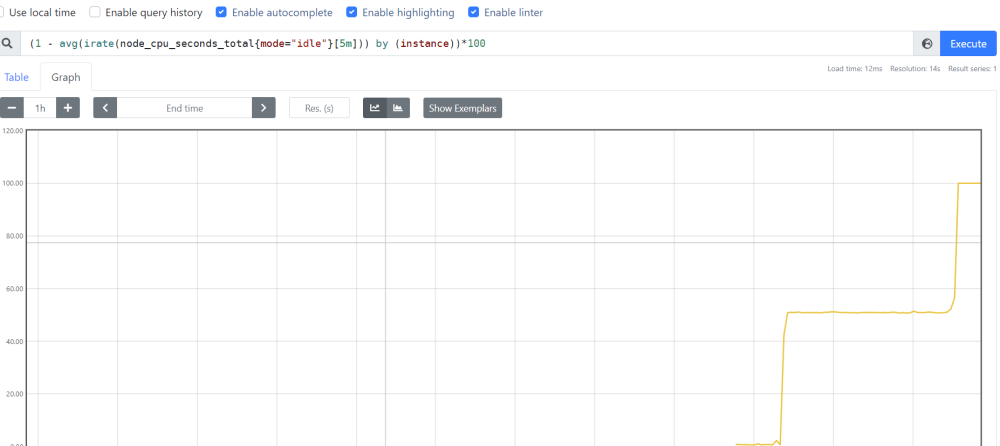
切换到Alerts标签http://127.0.0.1:9090/alerts可以查看当前告警的活动状态。



此时，我们可以手动拉高系统的CPU使用率，验证Prometheus的告警流程，在主机上运行以下命令：

vim load\_cpu.sh  
  
#!/bin/bash   
while true; do   
 :   
done  
  
chmod +x load\_cpu.sh  
./load\_cpu.sh

运行命令后查看CPU使用率情况，如下图所示：



Prometheus首次检测到满足触发条件后，hostCpuUsageAlert显示由一条告警处于活动状态。由于告警规则中设置了1m的等待时间，当前告警状态为PENDING，如下图所示：

