Docker部署Prometheus使用Alertmanager邮件报警

# 方案概述：

鉴于云服务器节点异常，导致相关已部署上线服务不可用。因此设计一个监控方案，在节点宕机时通过邮件提醒研发运维人员及时处理异常。

方案采用Prometheus(一款开源的系统监控报警框架)通过端口（exporter）采集相关数据，并且通过定义好的rules产生新的时间序列数据或者发送警报给Alertmanager(处理prome端发来的警报的框架),Alertmanager则通过邮件方式发送警报提醒。

# Docker安装

1. Docker要求centos内核版本高于3.10，查看当前内核版本。

$ uame -r

1. 使用 root 权限登录 Centos。确保 yum 包更新到最新。

$ sudo yum update

1. 安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的。

$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2

1. 设置yum源。

$ sudo yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

1. 安装docker。

$ sudo yum install docker-ce #由于repo中默认只开启stable仓库，故这里安装的是最新稳定版

$ sudo yum install <FQPN> # 例如：sudo yum install docker-ce-17.12.0.ce

1. 启动并加入开机启动。

$ sudo systemctl start docker

$ sudo systemctl enable docker

1. 验证是否成功。

$ docker version

# Prometheus配置安装

本文直接用官网的docker image（prom/prometheus）启动一个 Prometheus server, 并配置相应的静态监控 targets，jobs 和 alert.rules 文件。启动 Prometheus 容器，并把服务绑定在本机的 9090 端口。

1. 当前目录（你的工作目录）创建 vi prometheus.yml配置文件。

# my global config #严格注意格式问题，比如冒号后要有空格 每行后不要有空格等等，否则会无法运行

global:

scrape\_interval: 5s

evaluation\_interval: 5s

external\_labels:

monitor: 'codelab-monitor'

# Alertmanager configuration

alerting:

alertmanagers:

- static\_configs:

- targets:

- 192.168.2.140:9093

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files: #报警规则

- '/etc/prometheus/alert.rules'

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'prometheus' #监控prometheus自己

scrape\_interval: 5s

static\_configs:

- targets: ['192.168.2.140:9090']

- job\_name: 'node' #监控node\_exporter和cAdvisor， 两个均为测试时使用，不用可去掉

scrape\_interval: 5s

static\_configs:

- targets: ['192.168.2.140:9100','192.168.2.140:8080']

- job\_name: 'ceph\_exporter' #监控四个ceph节点

scrape\_interval: 5s

static\_configs:

- targets: ['192.168.2.140:9128','192.168.2.141:9128','192.168.2.142:9128','192.168.2.143:9128']

1. 同目录创建vi alert.rules（报警规则）配置文件。

groups: #严格注意格式问题，

- name: alert-name

rules:

- alert: InstanceDown

expr: up == 0

for: 10s

labels:

severity: critical

annotations:

summary: "Instance {{$labels.instance }} down"

description: "{{$labels.instance }} of job {{$labels.job }} has been down for more than 10 s."

1. 两个配置文件配置完毕开始创建容器(注意-v 后的路径是自己的文件存放路径，：后路径不用改)。

$docker run -d -p 9090:9090 --name prometheus -v ~/prometheus.yml:/etc/prometheus/prometheus.yml -v ~/alert.rules:/etc/prometheus/alert.rules prom/prometheus

1. 查看容器是否创建成功，未成功大多数原因的配置问题（去掉 -d运行可输出错误原因）。

$docker ps -a

1. 成功启动后可在浏览器https://localhost:9090查看。

另外alert如下图说明 alert.rules配置成功。

Inactive：这里什么都没有发生。  


# Alertmanager配置安装

1. 在工作目录创建配置文件vi alertmanager.yml。

global: #注意格式每行后的空格，不要有

smtp\_smarthost: 'smtp.163.com:25' #163 smtp服务的端口号是25，其他邮箱自行查询

smtp\_from: 'xxxx@163.com' #alert使用发送警报的邮箱

smtp\_auth\_username: 'xxxxx163.com' #再次输入邮箱

smtp\_auth\_password: 'gao473897902' #邮箱授权码，授权码自行百度

route:

group\_by: ['alertname']

group\_wait: 10s #等待时间

group\_interval: 10s #发送间隔

repeat\_interval: 10m # 重复发送等待10m分钟再发

receiver: live-monitoring

receivers:

- name: 'live-monitoring'

email\_configs:

- to: 'xxxxx@qq.com' #接收报警邮件的邮箱

1. 运行alertmanager

$docker run -d -p 9093:9093 --name alertmanager -v ~/alertmanager.yml:/etc/alertmanager/alertmanager.yml prom/alertmanager

1. 查看容器是否创建成功,未成功大多数原因的配置问题（去掉 -d运行可输出错误原因）。

$docker ps -a

1. 成功启动后可在浏览器https://localhost:9093查看。

# 测试环境搭建

至此报警环境搭建完毕，下面搭建测试环境，测试报警功能是否可以正常使用（测试环境按需搭建，该环境仅为了演示）。

1. 使用docker安装node\_exporter,拉取镜像。

$ docker pull quay.io/prometheus/node-exporter

1. 运行docker。

$ docker run -d --net="host" --pid="host" --cap-add=SYS\_TIME quay.io/prometheus/node-exporter

1. 查看容器是否创建成功,未成功大多数原因的配置问题（去掉 -d运行可输出错误原因）。

$ docker ps -a

1. 成功启动后可在浏览器https://localhost:9100查看。
2. 成功后需要在prometheus.yml增加监测端口配置，上面已经配置。

- job\_name: 'node' #监控node\_exporter和cAdvisor， 两个均为测试时使用，不用可去掉

scrape\_interval: 5s

static\_configs:

- targets: ['192.168.2.140:9100']

1. 配置后重启prometheus。

$ docker rm -f prometheus

$ docker run -d -p 9090:9090 --name prometheus -v ~/prometheus.yml:/etc/prometheus/prometheus.yml -v ~/alert.rules:/etc/prometheus/alert.rules prom/prometheus

1. 查看http://192.168.2.140:9090/targets。说明配置成功。

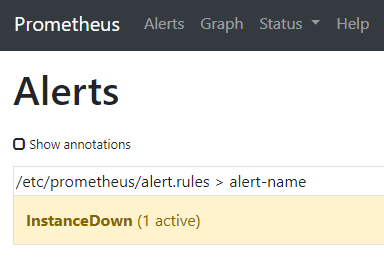


1. 模拟宕机（停止node\_exporter）。

$ docker stop (node\_exporterID)

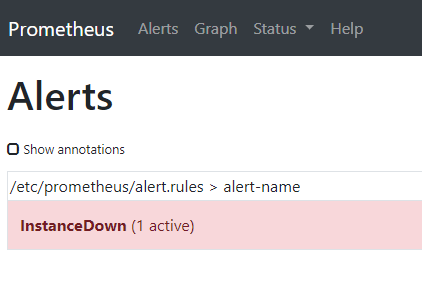
1. 等待数秒（具体时间可通过prometheus和alert配置文件调节）出错。

Pending：客户端告诉我们这个警报必须被触发。然而，警报可以被分组、压抑/抑制或者静默/静音。一旦所有的验证都通过了，我们就转到Firing。

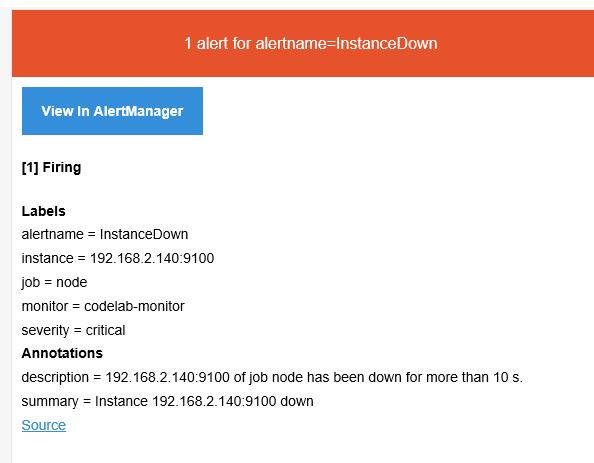




Firing：警报发送到Notification Pipeline，它将联系警报的所有接收者。然后客户端告诉我们警报解除，所以转换到状Inactive状态。



发送邮件



# 配置文件规则：

# Prometheus配置规则

一.参数的类型定义(通用占位符定义如下)

1. <boolean>：true false

2. <duration>：与正则表达式[0-9]+(ms|[smhdwy])匹配的持续时间

3. <labelname>：与正则表达式匹配的字符串[a-zA-Z\_][a-zA-Z0-9\_]\*

4. <labelvalue>：一个unicode字符串

5. <filename>：当前工作目录中的有效路径

6. <host>：一个有效的字符串，由一个主机名或IP后跟一个可选的端口号组成

7. <path>：有效的URL路径

8. <scheme>：可以取值为http或https的字符串

9. <string>：一个常规字符串

10. <secret>：一个常用的密码字符串，例如密码

11. <tmpl\_string>：在使用前被模板扩展的字符串

全局配置指定在所有其他配置上下文中有效的参数。 它们也作为其他配置部分的默认值。

global:

# How frequently to scrape targets by default.

[ scrape\_interval: <duration> | default = 1m ]

# How long until a scrape request times out.

[ scrape\_timeout: <duration> | default = 10s ]

# How frequently to evaluate rules.

[ evaluation\_interval: <duration> | default = 1m ]

# The labels to add to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

[ <labelname>: <labelvalue> ... ]

# Rule files specifies a list of globs. Rules and alerts are read from

# all matching files.搜集的规则文件

rule\_files:

[ - <filepath\_glob> ... ]

# A list of scrape configurations.

scrape\_configs:

[ - <scrape\_config> ... ]

# Alerting specifies settings related to the Alertmanager.

alerting:

alert\_relabel\_configs:

[ - <relabel\_config> ... ]

alertmanagers:

[ - <alertmanager\_config> ... ]

# Settings related to the remote write feature.

remote\_write:

[ - <remote\_write> ... ]

# Settings related to the remote read feature.

remote\_read:

[ - <remote\_read> ... ]

<scrape\_config>:定义收集规则。 在一般情况下，一个scrape配置指定一个job。 在高级配置中，这可能会改变。

relabel\_configs允许在抓取之前对任何目标及其标签进行修改。

目标可以通过static\_configs参数进行静态配置，也可以使用其中一种受支持的服务发现机制进行动态发现。

static\_configs

# The targets specified by the static config.

targets:

[ - '<host>' ]

# Labels assigned to all metrics scraped from the targets.

labels:

[ <labelname>: <labelvalue> ... ]

# The job name assigned to scraped metrics by default.

job\_name: <job\_name>

# How frequently to scrape targets from this job.

[ scrape\_interval: <duration> | default = <global\_config.scrape\_interval> ]

# Per-scrape timeout when scraping this job.

[ scrape\_timeout: <duration> | default = <global\_config.scrape\_timeout> ]

# The HTTP resource path on which to fetch metrics from targets.

[ metrics\_path: <path> | default = /metrics ]

# honor\_labels controls how Prometheus handles conflicts between labels that are

# already present in scraped data and labels that Prometheus would attach

# server-side ("job" and "instance" labels, manually configured target

# labels, and labels generated by service discovery implementations).

#

# If honor\_labels is set to "true", label conflicts are resolved by keeping label

# values from the scraped data and ignoring the conflicting server-side labels.

#

# If honor\_labels is set to "false", label conflicts are resolved by renaming

# conflicting labels in the scraped data to "exported\_<original-label>" (for

# example "exported\_instance", "exported\_job") and then attaching server-side

# labels. This is useful for use cases such as federation, where all labels

# specified in the target should be preserved.

#

# Note that any globally configured "external\_labels" are unaffected by this

# setting. In communication with external systems, they are always applied only

# when a time series does not have a given label yet and are ignored otherwise.

[ honor\_labels: <boolean> | default = false ]

# Configures the protocol scheme used for requests.

[ scheme: <scheme> | default = http ]

# Optional HTTP URL parameters.

params:

[ <string>: [<string>, ...] ]

# Sets the `Authorization` header on every scrape request with the

# configured username and password.

basic\_auth:

[ username: <string> ]

[ password: <secret> ]

# Sets the `Authorization` header on every scrape request with

# the configured bearer token. It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: <secret> ]

# Sets the `Authorization` header on every scrape request with the bearer token

# read from the configured file. It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: /path/to/bearer/token/file ]

# Configures the scrape request's TLS settings.

tls\_config:

[ <tls\_config> ]

# Optional proxy URL.

[ proxy\_url: <string> ]

# List of Azure service discovery configurations.

azure\_sd\_configs:

[ - <azure\_sd\_config> ... ]

# List of Consul service discovery configurations.

consul\_sd\_configs:

[ - <consul\_sd\_config> ... ]

# List of DNS service discovery configurations.

dns\_sd\_configs:

[ - <dns\_sd\_config> ... ]

# List of EC2 service discovery configurations.

ec2\_sd\_configs:

[ - <ec2\_sd\_config> ... ]

# List of OpenStack service discovery configurations.

openstack\_sd\_configs:

[ - <openstack\_sd\_config> ... ]

# List of file service discovery configurations.

file\_sd\_configs:

[ - <file\_sd\_config> ... ]

# List of GCE service discovery configurations.

gce\_sd\_configs:

[ - <gce\_sd\_config> ... ]

# List of Kubernetes service discovery configurations.

kubernetes\_sd\_configs:

[ - <kubernetes\_sd\_config> ... ]

# List of Marathon service discovery configurations.

marathon\_sd\_configs:

[ - <marathon\_sd\_config> ... ]

# List of AirBnB's Nerve service discovery configurations.

nerve\_sd\_configs:

[ - <nerve\_sd\_config> ... ]

# List of Zookeeper Serverset service discovery configurations.

serverset\_sd\_configs:

[ - <serverset\_sd\_config> ... ]

# List of Triton service discovery configurations.

triton\_sd\_configs:

[ - <triton\_sd\_config> ... ]

# List of labeled statically configured targets for this job.

static\_configs:

[ - <static\_config> ... ]

# List of target relabel configurations.

relabel\_configs:

[ - <relabel\_config> ... ]

# List of metric relabel configurations.

metric\_relabel\_configs:

[ - <relabel\_config> ... ]

# Per-scrape limit on number of scraped samples that will be accepted.

# If more than this number of samples are present after metric relabelling

# the entire scrape will be treated as failed. 0 means no limit.

[ sample\_limit: <int> | default = 0 ]

<tls\_config>配置详细信息

# CA certificate to validate API server certificate with.

[ ca\_file: <filename> ]

# Certificate and key files for client cert authentication to the server.

[ cert\_file: <filename> ]

[ key\_file: <filename> ]

# ServerName extension to indicate the name of the server.

# http://tools.ietf.org/html/rfc4366#section-3.1

[ server\_name: <string> ]

# Disable validation of the server certificate.

[ insecure\_skip\_verify: <boolean> ]

# Alermanager配置规则

1. route属性用来设置报警的分发策略，它是一个树状结构，按照深度优先从左向右的顺序进行匹配。
2. Alert是alertmanager接收到的报警，具有相同Lables的Alert（key和value都相同）才会被认为是同一种。在prometheus rules文件配置的一条规则可能会产生多种报警。
3. alertmanager会根据group\_by配置将Alert分组。如下规则，当go\_goroutines等于4时会收到三条报警，alertmanager会将这三条报警分成两组向receivers发出通知。
4. 主要处理流程：
5. 接收到Alert，根据labels判断属于哪些Route（可存在多个Route，一个Route有多个Group，一个Group有多个Alert）。
6. 将Alert分配到Group中，没有则新建Group。
7. 新的Group等待group\_wait指定的时间（等待时可能收到同一Group的Alert），根据resolve\_timeout判断Alert是否解决，然后发送通知。
8. 已有的Group等待group\_interval指定的时间，判断Alert是否解决，当上次发送通知到现在的间隔大于repeat\_interval或者Group有更新时会发送通知

global:

# ResolveTimeout is the time after which an alert is declared resolved

# if it has not been updated.

resolve\_timeout: 5m

# The smarthost and SMTP sender used for mail notifications.

smtp\_smarthost: 'xxxxx'

smtp\_from: 'xxxxxxx'

smtp\_auth\_username: 'xxxxx'

smtp\_auth\_password: 'xxxxxx'

# The API URL to use for Slack notifications.

slack\_api\_url: 'https://hooks.slack.com/services/some/api/token'

# # The directory from which notification templates are read.

templates:

- '\*.tmpl'

# The root route on which each incoming alert enters.

route:

# The labels by which incoming alerts are grouped together. For example,

# multiple alerts coming in for cluster=A and alertname=LatencyHigh would

# be batched into a single group.

group\_by: ['alertname', 'cluster', 'service']

# When a new group of alerts is created by an incoming alert, wait at

# least 'group\_wait' to send the initial notification.

# This way ensures that you get multiple alerts for the same group that start

# firing shortly after another are batched together on the first

# notification.

group\_wait: 30s

# When the first notification was sent, wait 'group\_interval' to send a batch

# of new alerts that started firing for that group.

group\_interval: 5m

# If an alert has successfully been sent, wait 'repeat\_interval' to

# resend them.

#repeat\_interval: 1m

repeat\_interval: 15m

# A default receiver

# If an alert isn't caught by a route, send it to default.

receiver: default

# All the above attributes are inherited by all child routes and can

# overwritten on each.

# The child route trees.

routes:

- match:

severity: critical

receiver: email\_alert

receivers:

- name: 'default'

email\_configs:

- to : 'yi.hu@dianrong.com'

send\_resolved: true

- name: 'email\_alert'

email\_configs:

- to : 'yi.hu@dianrong.com'

send\_resolved: true