# 2@Prometheus监控 - Alertmanager报警模块

- y\_xiao\_的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/y\_xiao\_/article/details/50818451

**Overview**

[Alertmanager](http://prometheus.io/docs/alerting/overview/)与Prometheus是相互分离的两个部分。Prometheus服务器根据报警规则将警报发送给Alertmanager，然后Alertmanager将silencing、inhibition、aggregation等消息通过电子邮件、PaperDuty和HipChat发送通知。

设置警报和通知的主要步骤：

* 安装配置Alertmanager
* 配置Prometheus通过-alertmanager.url标志与Alertmanager通信
* 在Prometheus中创建告警规则

**Alertmanager简介及机制**

Alertmanager处理由类似Prometheus服务器等客户端发来的警报，之后需要删除重复、分组，并将它们通过路由发送到正确的接收器，比如电子邮件、Slack等。Alertmanager还支持沉默和警报抑制的机制。

**分组**

分组是指当出现问题时，Alertmanager会收到一个单一的通知，而当系统宕机时，很有可能成百上千的警报会同时生成，这种机制在较大的中断中特别有用。

例如，当数十或数百个服务的实例在运行，网络发生故障时，有可能服务实例的一半不可达[**数据库**](http://lib.csdn.net/base/mysql)。在告警规则中配置为每一个服务实例都发送警报的话，那么结果是数百警报被发送至Alertmanager。

但是作为用户只想看到单一的报警页面，同时仍然能够清楚的看到哪些实例受到影响，因此，人们通过配置Alertmanager将警报分组打包，并发送一个相对看起来紧凑的通知。

分组警报、警报时间，以及接收警报的receiver是在配置文件中通过路由树配置的。

**抑制**

抑制是指当警报发出后，停止重复发送由此警报引发其他错误的警报的机制。

例如，当警报被触发，通知整个集群不可达，可以配置Alertmanager忽略由该警报触发而产生的所有其他警报，这可以防止通知数百或数千与此问题不相关的其他警报。

抑制机制可以通过Alertmanager的配置文件来配置。

**沉默**

沉默是一种简单的特定时间静音提醒的机制。一种沉默是通过匹配器来配置，就像路由树一样。传入的警报会匹配RE，如果匹配，将不会为此警报发送通知。

沉默机制可以通过Alertmanager的Web页面进行配置。

**Alertmanager的配置**

Alertmanager通过命令行flag和一个配置文件进行配置。命令行flag配置不变的系统参数、配置文件定义的禁止规则、通知路由和通知接收器。

要查看所有可用的命令行flag，运行alertmanager -h。

Alertmanager在运行时加载配置，如果不能很好的形成新的配置，更改将不会被应用，并记录错误。

**配置文件**

要指定加载的配置文件，需要使用-config.file标志。该文件使用YAML来完成，通过下面的描述来定义。括号内的参数是可选的，对于非列表的参数的值设置为指定的缺省值。

global:

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

# if it has not been updated.

[ resolve\_timeout: <duration> | default = 5m ]

# The default SMTP From header field.

[ smtp\_from: <tmpl\_string> ]

# The default SMTP smarthost used for sending emails.

[ smtp\_smarthost: <string> ]

# The API URL to use for Slack notifications.

[ slack\_api\_url: <string> ]

[ pagerduty\_url: <string> | default = "https://events.pagerduty.com/generic/2010-04-15/create\_event.json" ]

[ opsgenie\_api\_host: <string> | default = "https://api.opsgenie.com/" ]

# Files from which custom notification template definitions are read.

# The last component may use a wildcard matcher, e.g. 'templates/\*.tmpl'.

templates:

[ - <filepath> ... ]

# The root node of the routing tree.

route: <route>

# A list of notification receivers.

receivers:

- <receiver> ...

# A list of inhibition rules.

inhibit\_rules:

[ - <inhibit\_rule> ... ]

**路由 route**

路由块定义了路由树及其子节点。如果没有设置的话，子节点的可选配置参数从其父节点继承。

每个警报进入配置的路由树的顶级路径，顶级路径必须匹配所有警报（即没有任何形式的匹配）。然后匹配子节点。如果continue的值设置为false，它在匹配第一个孩子后就停止；如果在子节点匹配，continue的值为true，警报将继续进行后续兄弟姐妹的匹配。如果警报不匹配任何节点的任何子节点（没有匹配的子节点，或不存在），该警报基于当前节点的配置处理。

**路由配置格式**

[ receiver: <string> ]

[ group\_by: '[' <labelname>, ... ']' ]

# Whether an alert should continue matching subsequent sibling nodes.

[ continue: <boolean> | default = false ]

# A set of equality matchers an alert has to fulfill to match the node.

match:

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

# A set of regex-matchers an alert has to fulfill to match the node.

match\_re:

[ <labelname>: <regex>, ... ]

# How long to initially wait to send a notification for a group

# of alerts. Allows to wait for an inhibiting alert to arrive or collect

# more initial alerts for the same group. (Usually ~0s to few minutes.)

[ group\_wait: <duration> ]

# How long to wait before sending notification about new alerts that are

# in are added to a group of alerts for which an initial notification

# has already been sent. (Usually ~5min or more.)

[ group\_interval: <duration> ]

# How long to wait before sending a notification again if it has already

# been sent successfully for an alert. (Usually ~3h or more).

[ repeat\_interval: <duration> ]

# Zero or more child routes.

routes:

[ - <route> ... ]

示例：

# The root route with all parameters, which are inherited by the child

# routes if they are not overwritten.

route:

receiver: 'default-receiver'

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 4h

group\_by: [cluster, alertname]

# All alerts that do not match the following child routes

# will remain at the root node and be dispatched to 'default-receiver'.

routes:

# All alerts with service=mysql or service=cassandra

# are dispatched to the database pager.

- receiver: 'database-pager'

group\_wait: 10s

match\_re:

service: mysql|cassandra

# All alerts with the team=frontend label match this sub-route.

# They are grouped by product and environment rather than cluster

# and alertname.

- receiver: 'frontend-pager'

group\_by: [product, environment]

match:

team: frontend

**抑制规则 inhibit\_rule**

抑制规则，是存在另一组匹配器匹配的情况下，静音其他被引发警报的规则。这两个警报，必须有一组相同的标签。

**抑制配置格式**

# Matchers that have to be fulfilled in the alerts to be muted.

target\_match:

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

target\_match\_re:

[ <labelname>: <regex>, ... ]

# Matchers for which one or more alerts have to exist for the

# inhibition to take effect.

source\_match:

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

source\_match\_re:

[ <labelname>: <regex>, ... ]

# Labels that must have an equal value in the source and target

# alert for the inhibition to take effect.

[ equal: '[' <labelname>, ... ']' ]

**接收器 receiver**

顾名思义，警报接收的配置。

**通用配置格式**

# The unique name of the receiver.

name: <string>

# Configurations for several notification integrations.

email\_configs:

[ - <email\_config>, ... ]

pagerduty\_configs:

[ - <pagerduty\_config>, ... ]

slack\_config:

[ - <slack\_config>, ... ]

opsgenie\_configs:

[ - <opsgenie\_config>, ... ]

webhook\_configs:

[ - <webhook\_config>, ... ]

**邮件接收器 email\_config**

# Whether or not to notify about resolved alerts.

[ send\_resolved: <boolean> | default = false ]

# The email address to send notifications to.

to: <tmpl\_string>

# The sender address.

[ from: <tmpl\_string> | default = global.smtp\_from ]

# The SMTP host through which emails are sent.

[ smarthost: <string> | default = global.smtp\_smarthost ]

# The HTML body of the email notification.

[ html: <tmpl\_string> | default = '{{ template "email.default.html" . }}' ]

# Further headers email header key/value pairs. Overrides any headers

# previously set by the notification implementation.

[ headers: { <string>: <tmpl\_string>, ... } ]

**Slack接收器 slack\_config**

# Whether or not to notify about resolved alerts.

[ send\_resolved: <boolean> | default = true ]

# The Slack webhook URL.

[ api\_url: <string> | default = global.slack\_api\_url ]

# The channel or user to send notifications to.

channel: <tmpl\_string>

# API request data as defined by the Slack webhook API.

[ color: <tmpl\_string> | default = '{{ if eq .Status "firing" }}danger{{ else }}good{{ end }}' ]

[ username: <tmpl\_string> | default = '{{ template "slack.default.username" . }}'

[ title: <tmpl\_string> | default = '{{ template "slack.default.title" . }}' ]

[ title\_link: <tmpl\_string> | default = '{{ template "slack.default.titlelink" . }}' ]

[ pretext: <tmpl\_string> | default = '{{ template "slack.default.pretext" . }}' ]

[ text: <tmpl\_string> | default = '{{ template "slack.default.text" . }}' ]

[ fallback: <tmpl\_string> | default = '{{ template "slack.default.fallback" . }}' ]

**Webhook接收器 webhook\_config**

# Whether or not to notify about resolved alerts.

[ send\_resolved: <boolean> | default = true ]

# The endpoint to send HTTP POST requests to.

url: <string>

Alertmanager会使用以下的格式向配置端点发送HTTP POST请求：

{

"version": "2",

"status": "<resolved|firing>",

"alerts": [

{

"labels": <object>,

"annotations": <object>,

"startsAt": "<rfc3339>",

"endsAt": "<rfc3339>"

},

...

]

}

**报警规则**

报警规则允许你定义基于Prometheus语言表达的报警条件，并发送报警通知到外部服务。

**定义报警规则**

报警规则通过以下格式定义：

ALERT <alert name>

IF <expression>

[ FOR <duration> ]

[ LABELS <label set> ]

[ ANNOTATIONS <label set> ]

FOR子句使得Prometheus等待第一个传进来的向量元素（例如高HTTP错误的实例），并计数一个警报。如果元素是active，但是没有firing的，就处于pending状态。

LABELS（标签）子句允许指定一组附加的标签附到警报上。现有的任何标签都会被覆盖，标签值可以被模板化。

ANNOTATIONS（注释）子句指定另一组未查明警报实例的标签，它们被用于存储更长的其他信息，例如警报描述或者链接，注释值可以被模板化。

**报警规则示例**

# Alert for any instance that is unreachable for >5 minutes.

ALERT InstanceDown

IF up == 0

FOR 5m

LABELS { severity = "page" }

ANNOTATIONS {

summary = "Instance {{ $labels.instance }} down",

description = "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes.",

}

# Alert for any instance that have a median request latency >1s.

ALERT APIHighRequestLatency

IF api\_http\_request\_latencies\_second{quantile="0.5"} > 1

FOR 1m

ANNOTATIONS {

summary = "High request latency on {{ $labels.instance }}",

description = "{{ $labels.instance }} has a median request latency above 1s (current value: {{ $value }}s)",

}

**发送警报通知**

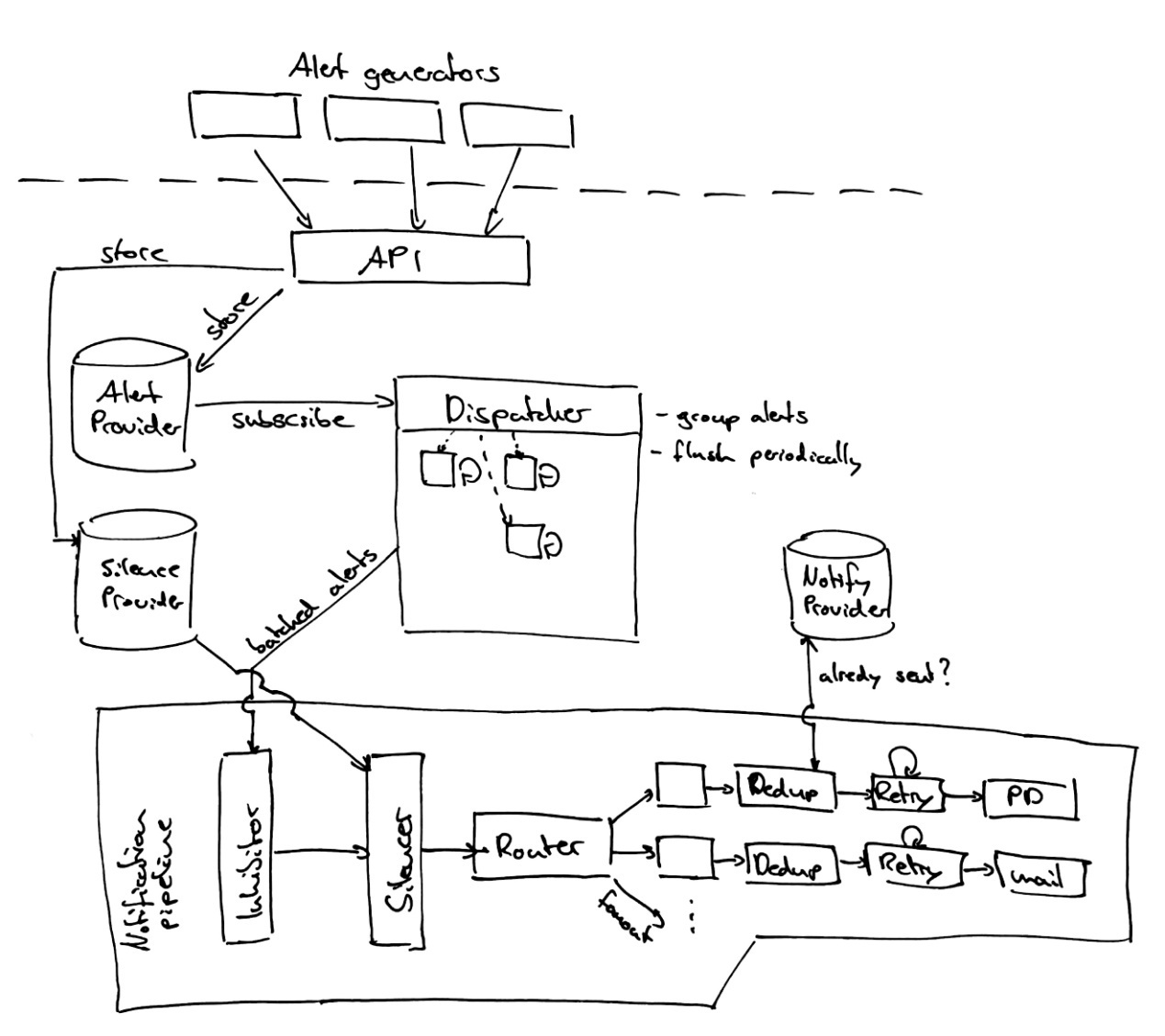
Prometheus可以周期性的发送关于警报状态的信息到Alertmanager实例，然后Alertmanager调度来发送正确的通知。该Alertmanager可以通过-alertmanager.url命令行flag来配置。

# 2@Prometheus with Alertmanager

http://www.songjiayang.com/technical/prometheus-with-alertmanager/

*强烈推荐使用官方提供的 alertmanager 实现告警通知*

20 Oct 2016 in [Technical](http://www.songjiayang.com/categories/index.html#technical)  2 minutes read



在文章 [How to monitor your system with prometheus](http://www.songjiayang.com/technical/prometheus-with-alertmanager/technical/how-to-monitor-your-system-with-prometheus/) 中介绍过 Prometheus 的安装教程，相信看过的小伙伴都已经开始使用了。但这样就够了吗，是否还缺少点什么？

对，没错，我们缺少告警 ！

图表再好看，也不会时时刻刻盯着啊，个人认为一个好的监控系统是能够自发告警，只有在有告警的时候，我们才登上去多喵几眼。

其实，Prometheus 官方提供了告警模块，那就是 [Alertmanager](https://github.com/prometheus/alertmanager), 不过因为代码解耦关系，它被单独剥离成一个独立项目了。

我是分割线

下面我们将完成安装和配置 Alertmanager, 并结合 Onealert 实现报警需求

环境准备

* 已安装 Prometheus Server (参考 [How to monitor your system with prometheus](http://www.songjiayang.com/technical/prometheus-with-alertmanager/technical/how-to-monitor-your-system-with-prometheus/))

**Step 1 — 安装 Alertmanager**

使用 wget 下载 Alertmanager 安装包

cd ~/Download

wget https://github.com/prometheus/alertmanager/releases/download/v0.4.2/alertmanager-0.4.2.linux-amd64.tar.gz

cd Prometheus

使用 tar 解压缩 alertmanager-0.4.2.linux-amd64.tar.gz

tar -xvzf ~/Download/alertmanager-0.4.2.linux-amd64.tar.gz

cd alertmanager-0.4.2.linux-amd64

当解压缩成功后，可以运行 version 检查运行环境是否正常

./alertmanager -version

如果你看到类似输出，表示你已安装成功:

alertmanager, version xxx (branch: master, revision: 9a5ab2fa63dd7951f4f202b0846d4f4d8e9615b0)

build user: root@2811d2f42616

build date: 20160902-15:34:07

go version: go1.6.3

**Step 2 — 修改 alertmanager 配置**

修改 simple.yml 文件，使用 Onealert 的 Prometheus 默认应用配置

**#simple.yml**

global:

resolve\_timeout: 1m

route:

group\_by: ['alertname']

group\_wait: 5s

group\_interval: 10s

repeat\_interval: 1h

receiver: 'webhook'

receivers:

- name: 'webhook'

webhook\_configs:

- url: 'http://api.110monitor.com/alert/api/event/prometheus/xxxxxxxxx'

send\_resolved: true

提示: 这里我们使用 Alertmanager 的 webhook\_configs，地址为 Onealert 一个 App

**Step 3 — 启动 Alertmanager**

使用 ./alertmanager -config.file simple.yml 运行 Alertmanager, 如果看到类似输出，表示运行成功

INFO[0000] Starting alertmanager (version=0.4.2, branch=master, revision=9a5ab2fa63dd7951f4f202b0846d4f4d8e9615b0) source=main.go:84

INFO[0000] Build context (go=go1.6.3, user=root@2811d2f42616, date=20160902-15:34:07) source=main.go:85

INFO[0000] Loading configuration file file=simple.yml source=main.go:156

INFO[0000] Listening on :9093 source=main.go:206

**Step 4 — 添加 Rules**

切换到 Prometheus Server 目录, 修改 prometheus.yml 文件，添加 rule\_files

**#prometheus.yml**

rule\_files:

- "first.rules"

创建 first.rules 文件, 并添加

**#first.rules**

ALERT InstanceStatus

IF up{job="node"} == 0

FOR 10s

LABELS {

instance = "",

}

ANNOTATIONS {

summary = "服务器 运行状态",

description = "服务器 已当机超过 20s"

}

使用命令 ./prometheus -alertmanager.url http://localhost:9093 重启 Promtheus Server

此时在浏览器中访问页面 http://localhost:9090/alerts，你将看到配置的所有 Rules



停掉 Node Exporter，隔一定时间，刷新该页面，你将看到



此时你会收到类似的告警邮件（Onealert 默认通知配置）



至此，我们已完成了使用 Alertmanager 来实现应用的告警通知。

结语：

Prometheus 代码是非常解耦的，我们可以使用官方的 Alertmanager 包，再结合 PromQL 强大能力，简单配置 Rules， 即满足常见告警需求，对于比较特殊的告警，则需要结合 Alertmanager 更加细致的配置实现。

我已仔细阅读代码，发现它在告警聚合，去重，以及配置不同发送频率，不同渠道都做的比较完备，以后会展开细讲。

# 3@promethus监控Ubuntu主机demo

http://blog.leanote.com/post/mozhata/promethus%E7%9B%91%E6%8E%A7Ubuntu%E4%B8%BB%E6%9C%BA

参考链接：[Monitoring linux stats with Prometheus.io](https://resin.io/blog/monitoring-linux-stats-with-prometheus-io/)

这个demo展示如何使用prometheus监控Ubuntu系统信息

大概流程大概是这个样子

1. node\_exporter监控主机信息并提供API接口
2. prometheus定时拉去node\_exporter的监控信息
3. prometheus定时检查报警条件是否触发，若触发则发送报警信息给Alertmanager处理
4. Alertmanager决定如何处理报警，比如忽略，发送邮件等

准备工作

* 安装[prometheus主程序](https://github.com/prometheus/prometheus)， 为了方便，这里直接用了docker镜像

1. docker pull prom/prometheus

如果要直接用二进制文件部署的话需要指定一些依赖文件，比如html模板，依赖库等， 具体可查看[prometheus项目的Dockerfile](https://github.com/prometheus/prometheus/blob/master/Dockerfile)

* 安装[alertmanager](https://github.com/prometheus/alertmanager/)用于报警服务
* 安装[node\_exporter](https://github.com/prometheus/node_exporter)用于监控主机的CPU 内存 磁盘状况等信息

创建目录prometheus\_demo,所有先关文件都放在这个目录内

1. mkdir -p /home/root/prometheus\_demo

* 创建prometheus.yml文件，用来配置prometheus
* 创建alert.rules文件，用来定义触发报警条件
* 创建alertmanager.yml 配置alertmanager

文件结构类似这个样子

1. ├── alertmanager.yml
2. ├── alert.rules
3. └── prometheus.yml

监控数据

这里使用node\_exporter作为exporters监控Ubuntu主机信息。   
默认端口为9100，也可以使用-web.listen-address ":<PORT>"参数修改，访问http://<your-device-ip>:9100/metrics查看node\_exporter的监控项

1. node\_exporter &

这里就是http://localhost://9100/metrics

有个问题， 因为prometheus主程序用的是docker，所以主程序无法直接访问到localhost这个地址，可以使用主机ip

配置**prometheus**

收集监控数据到**prometheus**

主要就是配置target然后加一些label， 大概像这个样子

1. *// prometheus.yml*
2. scrape\_configs:
3. - job\_name: "node"
4. static\_configs:
5. - targets: ["192.168.0.66:9100"]
6. labels:
7. device\_ID: "local"

其中192.168.0.66 是我的主机ip

设置警报触发条件

修改alert.rules 写入报警触发条件

1. *// alert.rules*
2. ALERT cpu\_threshold\_exceeded
3. IF (100 \* (1 - avg by(job)(irate(node\_cpu{mode='idle'}[5m])))) > 6
4. ANNOTATIONS {
5. summary = "Instance {{ $labels.instance }} CPU usage is dangerously high",
6. description = "This device's CPU usage has exceeded the threshold with a value of {{ $value }}.",
7. }
8. ALERT service\_down
9. IF up == 0
10. ANNOTATIONS {
11. summary = "Instance {{ $labels.instance }} is down",
12. }

其中，第一条：cpu\_threshold\_exceeded 当CPU利用率超过6%之后触发报警   
第二条：service\_down 当监控服务挂掉则报警，我们只有一个target， 把node\_exporter进程杀掉就会触发报警

把alert.rules中的规则应用到prometheus:

1. *// prometheus.yml*
2. rule\_files:
3. - "/conf/alert.rules"

这里的文件路径是/conf而不是上面提到的/home/root/prometheus\_demo，是因为这个目录是因为这里用的是docker， /conf是主机目录/home/root/prometheus\_demo在容器内的映射目录，后面会提到

配置**alertmanager**链接

1. *// prometheus.yml*
2. alerting:
3. alertmanagers:
4. - scheme: http
5. static\_configs:
6. - targets:
7. - "192.168.0.66:9093"

其中"192.168.0.66:9093"是alertmanager

prometheus.yml文件配置完之后是这个样子

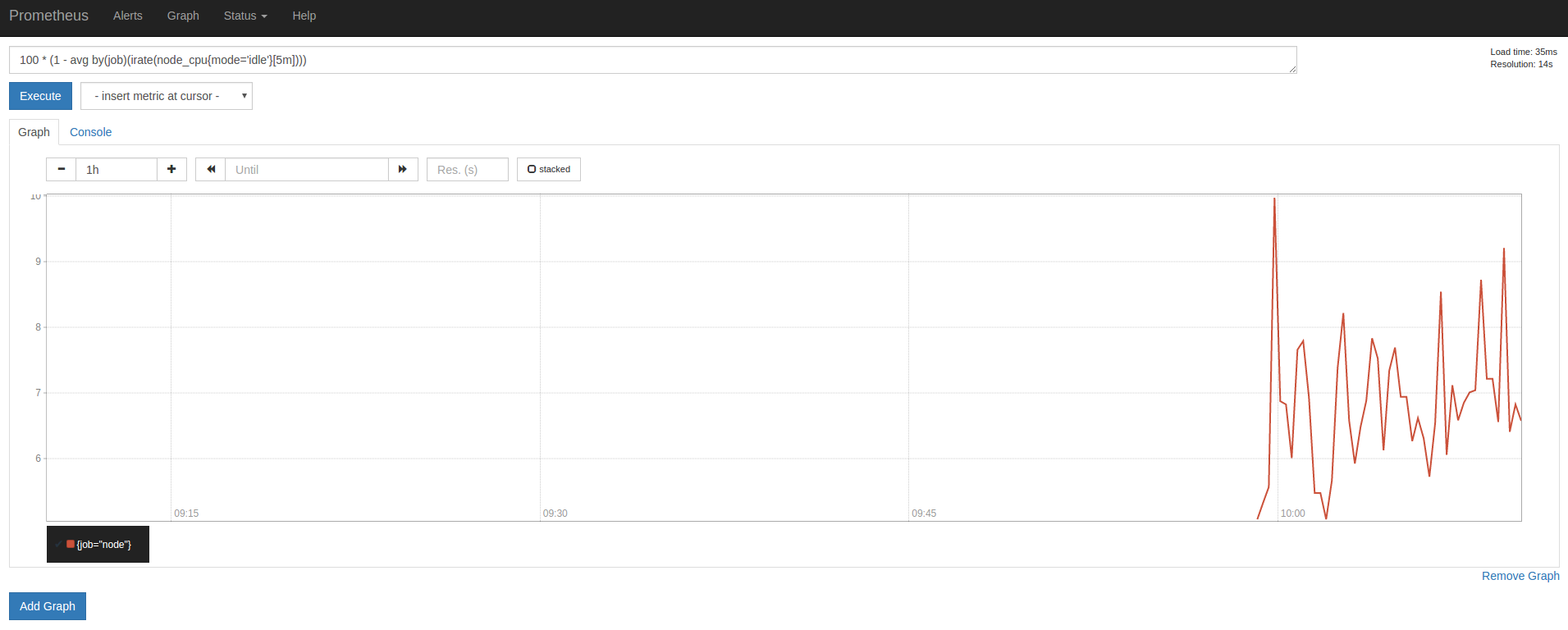
1. *# config for docker*
2. global:
3. scrape\_interval: 15s *# By default, scrape targets every 15 seconds.*
4. evaluation\_interval: 15s *# Evaluate rules every 15 seconds.*
5. *# Attach these extra labels to all timeseries collected by this Prometheus instance.*
6. external\_labels:
7. monitor: 'codelab-monitor'
8. rule\_files:
9. - "/conf/alert.rules"
10. scrape\_configs:
11. - job\_name: "node"
12. static\_configs:
13. - targets: ["192.168.0.66:9100"]
14. labels:
15. device\_ID: "local"
16. alerting:
17. alertmanagers:
18. - scheme: http
19. static\_configs:
20. - targets:
21. - "192.168.0.66:9093"

启动prometheus：

1. docker run --rm -p 9090:9090 \
2. -v /home/root/prometheus\_demo:/conf \
3. -d prom/prometheus \
4. -config.file=/conf/prometheus.yml \
5. -storage.local.path=/prometheus \
6. -web.console.libraries=/usr/share/prometheus/console\_libraries \
7. -web.console.templates=/usr/share/prometheus/consoles

打开http://localhost:9090/alerts查看定义的报警规则   


点击链接可以直接跳到graph界面



接下来配置Alertmanager

配置**Alertmanager**

这里设置的是邮件报警

1. *// alertmanager.yml*
2. global:
3. *# The smarthost and SMTP sender used for mail notifications.*
4. smtp\_smarthost: 'smtp.163.com:25'
5. smtp\_from: "sender@163.com"
6. smtp\_auth\_username: "sender@163.com"
7. smtp\_auth\_password: 'SENDERPWD'
8. *# The auth token for Hipchat.*
9. *# hipchat\_auth\_token: 'SENDERPWD'*
10. route:
11. group\_by: [Alertname]
12. *# Send all notifications to me.*
13. receiver: email-me
14. *# When a new group of alerts is created by an incoming alert, wait at*
15. *# least 'group\_wait' to send the initial notification.*
16. *# This way ensures that you get multiple alerts for the same group that start*
17. *# firing shortly after another are batched together on the first*
18. *# notification.*
19. group\_wait: 30s
20. *# When the first notification was sent, wait 'group\_interval' to send a batch*
21. *# of new alerts that started firing for that group.*
22. group\_interval: 5m
23. *# If an alert has successfully been sent, wait 'repeat\_interval' to*
24. *# resend them.*
25. repeat\_interval: 5m
26. receivers:
27. - name: email-me
28. email\_configs:
29. - to: "receiver@aliyun.com"

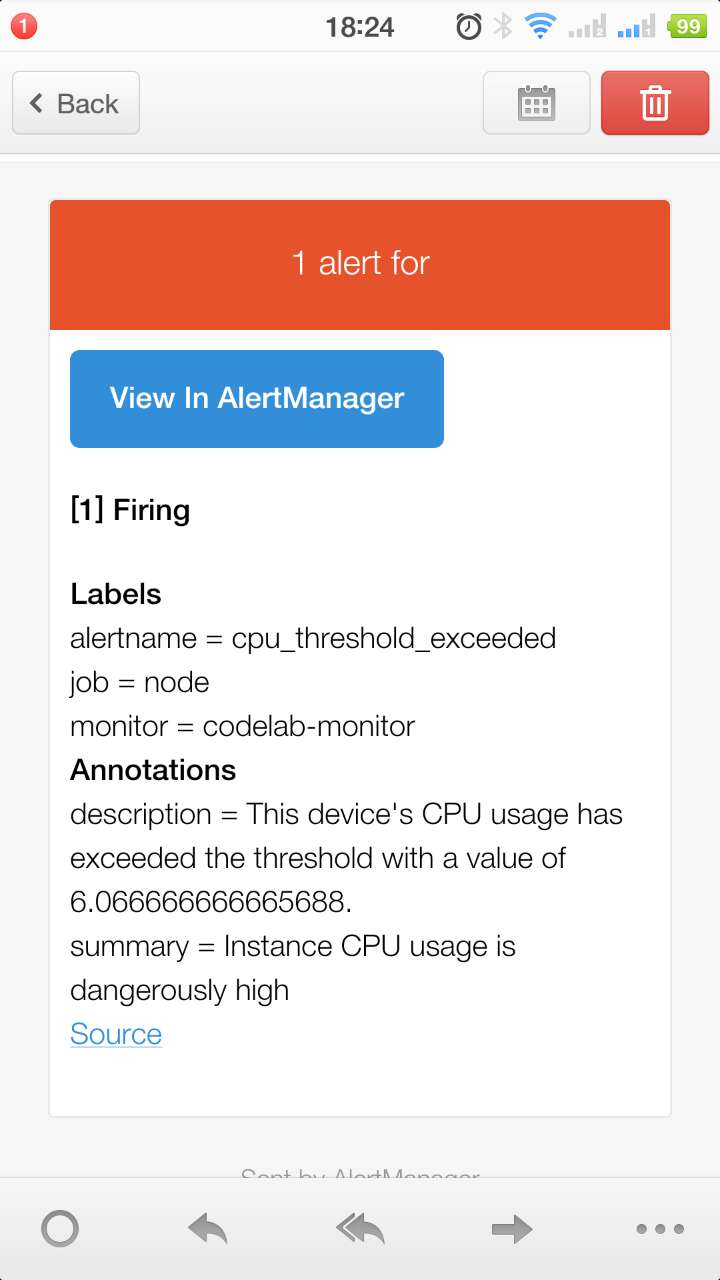
其中SENDERPWD是邮箱sender@163.com的密码

启动Alertmanager:

1. alertmanager -config.file=./alertmanager.yml &

当有报警需要处理的时候就会使用邮箱sender@163.com向邮箱receiver@aliyun.com发送邮件

像这样：



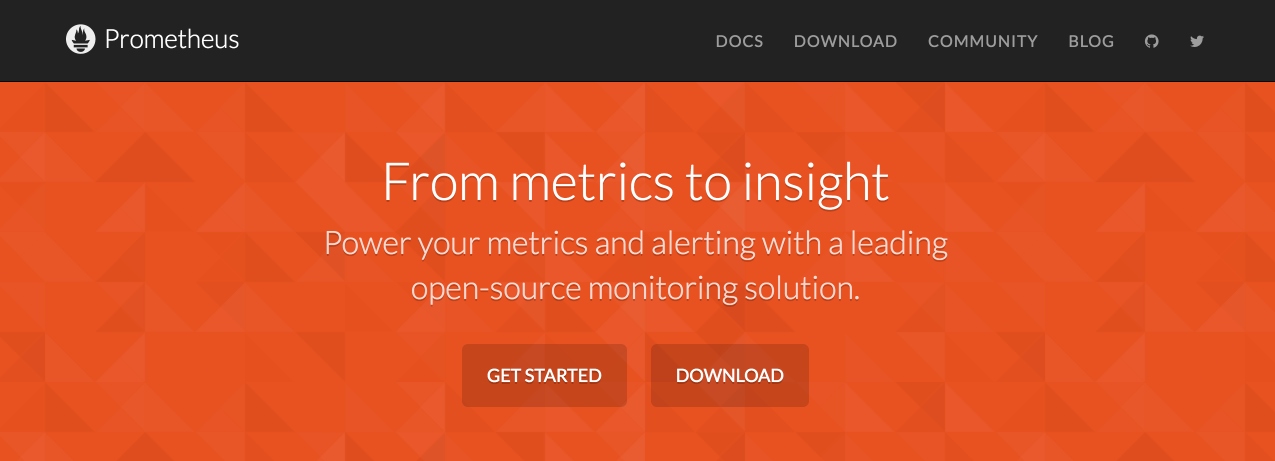
上一篇: [sublime 笔记](http://blog.leanote.com/post/mozhata/sublime-%E7%AC%94%E8%AE%B0)

# Monitoring linux stats with Prometheus.io

https://resin.io/blog/monitoring-linux-stats-with-prometheus-io/

Monitoring linux stats with Prometheus.io

This is the first of two tutorials on monitoring machine metrics of your device fleet with [Prometheus.io](https://prometheus.io/).



Prometheus is a tool, initially built by [soundcloud](https://soundcloud.com/) to monitor their servers, it is now open-source and completely community driven. It works by scraping "targets" which are endpoints that post key-attribute machine parseable data. Prometheus then stores each scrape as a frame in a time series database allowing you to query the database to execute graphs and other functions like alerts.

**tl;dr** This post runs through the configuration required to get prometheus.io running on a fleet of resin.io devices. [Skip ahead](https://resin.io/blog/monitoring-linux-stats-with-prometheus-io/#runningthedemo) if you'd like to go straight to deploying and running the demo.

Collecting

The first task is collecting the data we'd like to monitor and reporting it to a URL reachable by the Prometheus server. This is done by pluggable components which Prometheus calls exporters. We're going to use a common exporter called the node\_exporter which gathers Linux system stats like CPU, memory and disk usage. [Here](https://github.com/prometheus/node_exporter#collectors) is a full list of the stats the node\_exporter collects.

Once the exporter is running it'll host the parseable data on port 9100, this is configurable by passing the flag -web.listen-address ":<PORT>" when spawning the exporter.

Once running visit http://<your-device-ip>:9100/metrics, you'll see all the machine metrics from the node\_exporter in plain-text.

Scraping

We now need to make sense of the data being collected thats where the[Prometheus server](https://github.com/prometheus/prometheus) comes in. [prometheus.yml](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/prometheus.yml) holds the configuration that tells Prometheus how and which exporters to scrape. Let's go run through it and explain the configurations.

The scrape\_interval is the interval that Prometheus will scrape it's targetswhich are exporter endpoints. This will control the granularity of the time-series database. We have set it to scrape every 5s this is for demo purposes, usually, you’d use something like 60s.

global:

scrape\_interval: "5s"

Rule files specify a list of files from which rules are read, these rules and trigger webhooks to set alerts more on that a little later.

rule\_files:

- 'alert.rules'

The scrape\_configs allow you to set targets for each scrape job. We will give it a job\_name composed of the exporter name and a trimmed version of the [RESIN\_DEVICE\_UUID](http://docs.resin.io/management/devices/#identifying-devices) we then add target endpoints which is thenode\_exporter server we mentioned in the previous section. Then we add some resin specific labels to make identifying the target easier.

scrape\_configs:

- job\_name: "node"

static\_configs:

- targets:

- "localhost:9100"

labels:

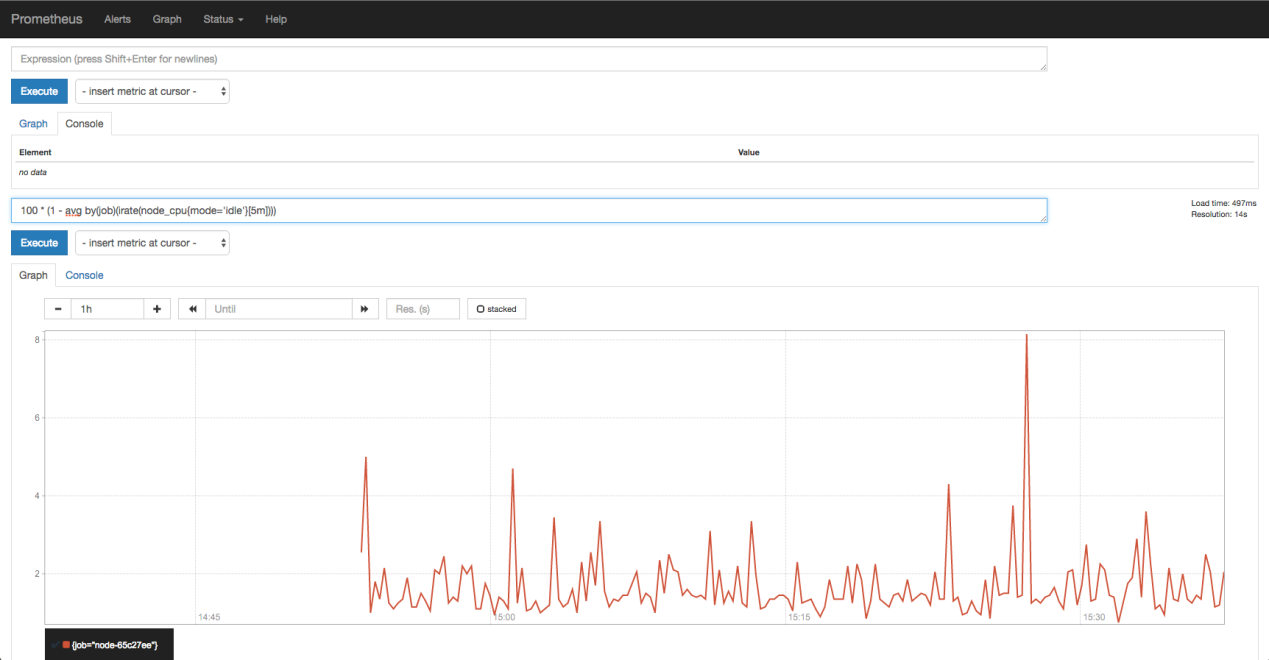
resin\_app: RESIN\_APP\_ID

resin\_device\_uuid: RESIN\_DEVICE\_UUID

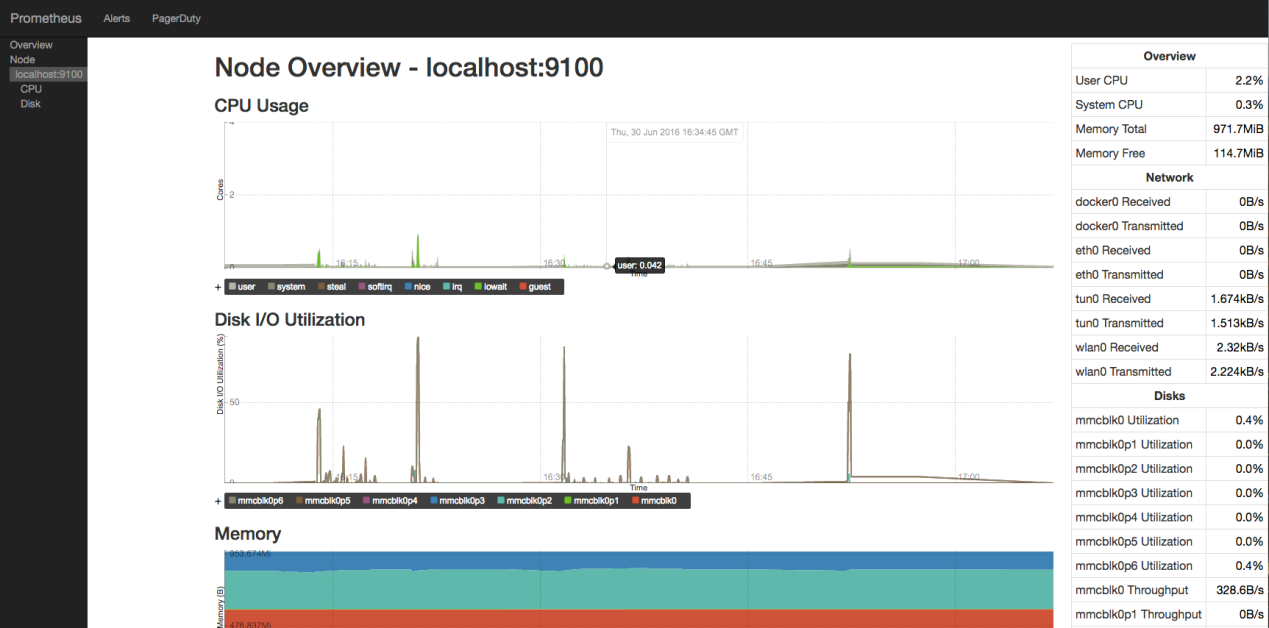
These labels are replaced at run time with real values from resin environment using the [config.sh](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/config.sh).

Once the Prometheus server is running, [activate](http://docs.resin.io/#/pages/management/devices.md#enable-public-device-url) and visit the device's public URL.

On the prometheus dashboard there won't be much to start let's add a graph. Select add graph and add100 \* (1 - avg by(instance)(irate(node\_cpu{job='node',mode='idle'}[5m]))) to the expression input. This query will find the average % CPU between the last two data points going back as far as 5 minutes, if no data point before that exists. Now we are able to query scraped data, great!



Prometheus also comes with a pre-configured console for node\_exporterjobs to view this visiting http://<your-device-url>/consoles/node.html. Voila!



Alerts

So now we have a way to collect data as well as query that data. Let's set up some alerts, these will query the scraped instances and send us an email if those queries evaluate to an undesirable value.

The alerting is handled by a separate component, the [alertmanager](https://github.com/prometheus/alertmanager/).

The first thing to do is create some rules that Prometheus can check after each scrape. We have defined several rules in [alert.rules](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/alert.rules) to get you started, let's take a look at one of them.

ALERT cpu\_threshold\_exceeded

IF (100 \* (1 - avg by(job)(irate(node\_cpu{mode='idle'}[5m])))) > THRESHOLD\_CPU

ANNOTATIONS {

summary = "Instance {{ $labels.instance }} CPU usage is dangerously high",

description = "This device's CPU usage has exceeded the threshold with a value of {{ $value }}.",

}

Here we define the alert and pass it the query we mentioned earlier as an if statement. If the statement finds that the value exceeds ourTHRESHOLD\_CPU it will trigger the alert. We have also added ANNOTATIONS to make the alert more descriptive.

We then need to configure the Alertmanager to handle the actions required when an alert is fired. The configuration can be found in[alertmanager.yml](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/alertmanager.yml). The Alertmanager isn't a full SMTP server itself, however it can pass on emails to something like Gmail which can send them on our behalf.

route:

group\_by: [Alertname]

# Send all notifications to me.

receiver: email-me

# 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: 3h

templates:

- '/etc/ALERTMANAGER\_PATH/default.tmpl'

receivers:

- name: email-me

email\_configs:

- to: GMAIL\_ACCOUNT

from: GMAIL\_ACCOUNT

smarthost: smtp.gmail.com:587

html: '{{ template "email.default.html" . }}'

auth\_username: "GMAIL\_ACCOUNT"

auth\_identity: "GMAIL\_ACCOUNT"

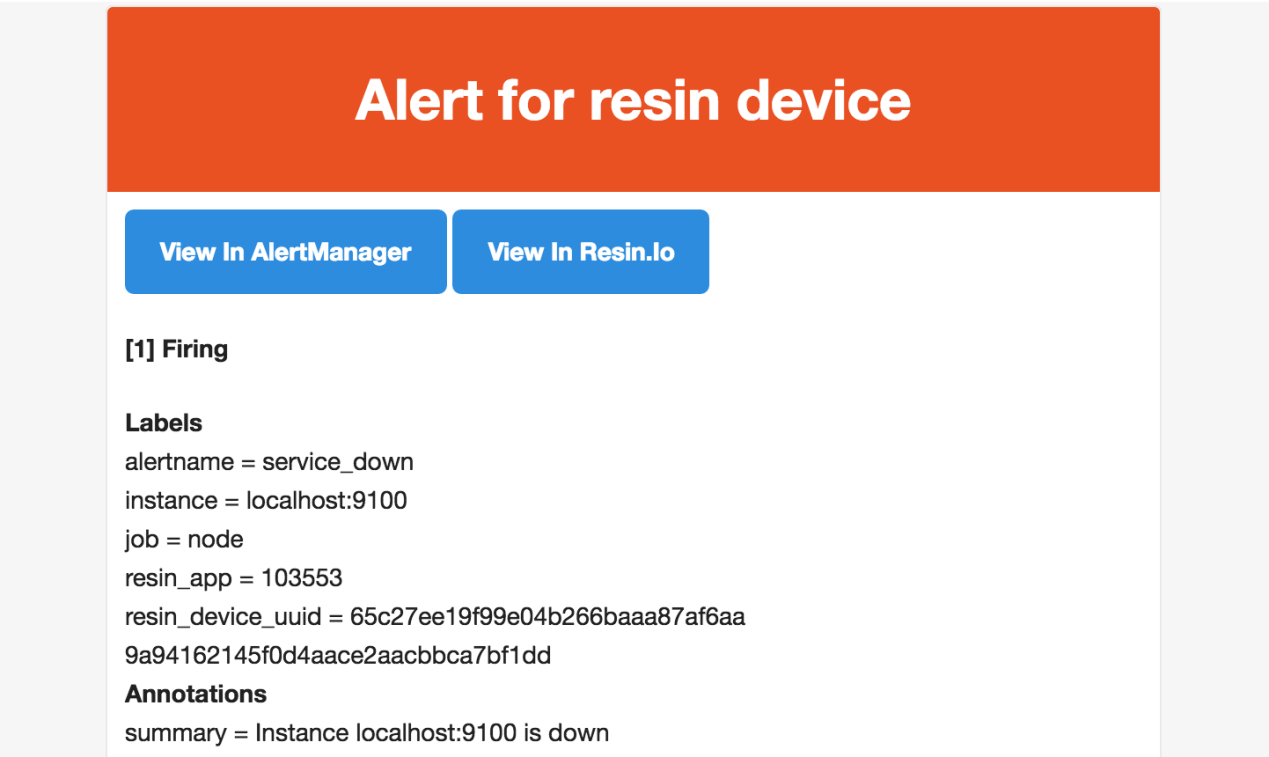
auth\_password: "GMAIL\_AUTH\_TOKEN"

Here we catch all alerts and send the to the receiver email-me.   
Take note of templates: - '/etc/ALERTMANAGER\_PATH/default.tmpl' This overrides the default templates used for notifications with our own which you can see in [default.tmpl](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/default.tmpl).

Testing the alerts

Visit http://<your-device-url>/alerts, you'll see a couple alerts we defined earlier in [alert.rules](https://github.com/resin-io-projects/resin-prometheus/blob/master/config/alert.rules) The easiest one to test is the first, service\_down, this alert is triggered when the targets (the instances prometheus server is scraping) drops to zero. Because we only have one target instance (node\_exporter) we may simply kill that process to trigger the alert.

Using the [resin web terminal](http://docs.resin.io/runtime/terminal/) run $ pkill node\_exporter. If you then refresh http://<your-device-url>/alerts you'll see the alert is firing. You will also get an email with the alert description and which device it is affecting.



Running the demo

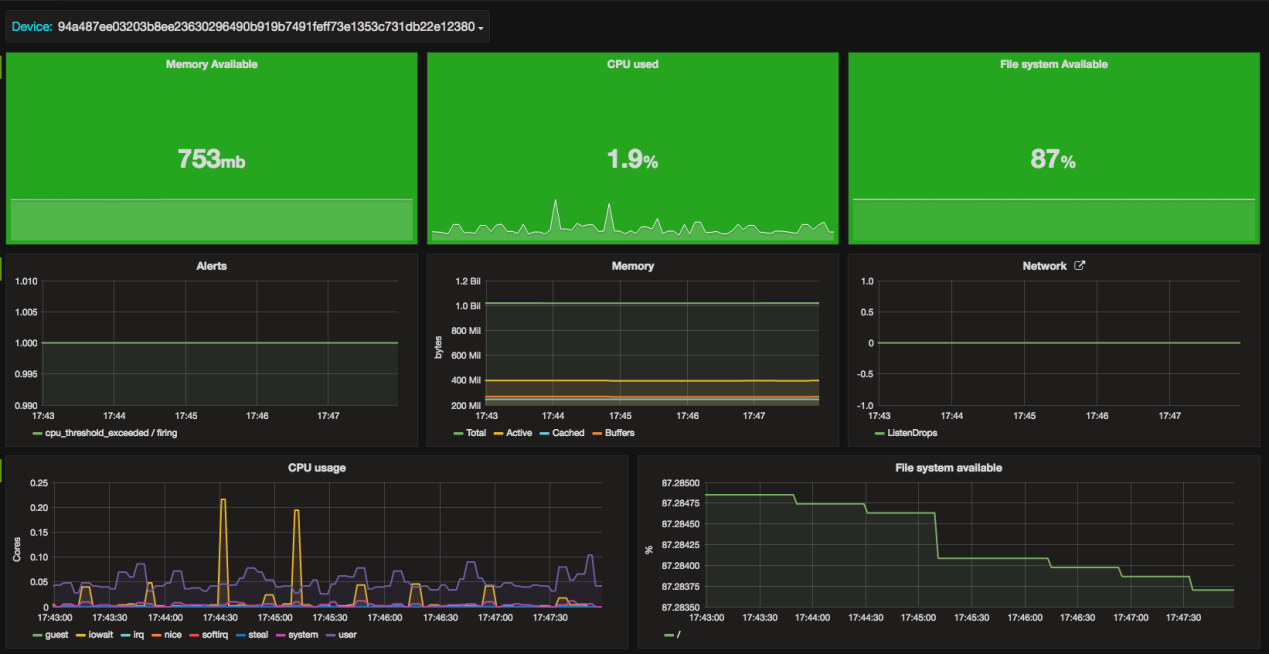
Find the code [here](https://github.com/resin-io-projects/resin-prometheus).

1. [Provision you're device(s) with resin.io](http://docs.resin.io/raspberrypi/nodejs/getting-started/)
2. git clone https://github.com/resin-io-projects/resin-prometheus && cd resin-prometheus
3. git add remote resin <your-resin-app-endpoint>
4. git push resin master
5. Add the following variables as [Application-wide environment variables](http://docs.resin.io/management/env-vars/#application-wide)

| **Key** | **Value** | **Default** | **Required** |
| --- | --- | --- | --- |
| GMAIL\_ACCOUNT | your Gmail email |  | \* |
| GMAIL\_AUTH\_TOKEN | you Gmail password or auth token |  | \* |
| THRESHOLD\_CPU | max % of CPU in use | 50 |  |
| THRESHOLD\_FS | min % of filesystem available | 50 |  |
| THRESHOLD\_MEM | min MB of mem available | 800 |  |
| LOCAL\_STORAGE\_RETENTION | Period of data retention | 360h0m0s |  |

Coming up

This demo provides some basic fleet monitoring, but there it doesn't provide a complete view of your entire fleets statistics in a single dashboard. In the next post we'll connect all the devices to a central[grafana dashboard](http://grafana.org/) for fleet-wide, as well as a device specific views - stay tuned!



Any questions? - ask us on [gitter](https://gitter.im/resin-io/chat)!

# Prometheus Alertmanager with slack receiver

http://www.songjiayang.com/technical/prometheus-alert-slack-receiver/

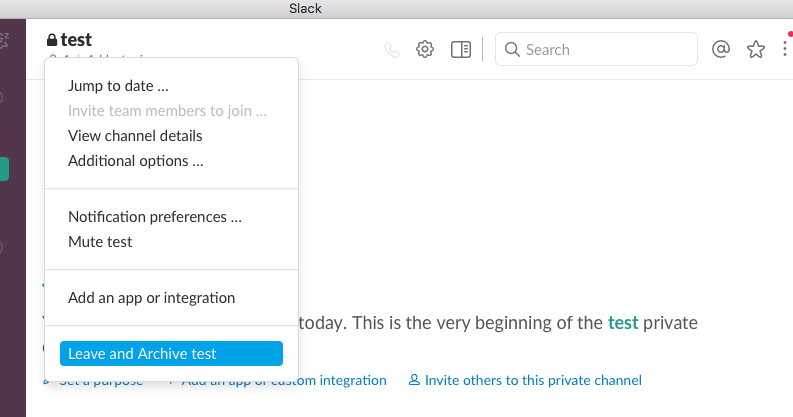
在过去一篇文章 [Prometheus With Alertmanager](http://www.songjiayang.com/technical/prometheus-with-alertmanager) 中， 已介绍了 prometheus 的告警模块，Alertmanager 用法；今天我们将一起学习，使用 slack 接收告警通知，让咱们的运维看上去高大上，我们想做：

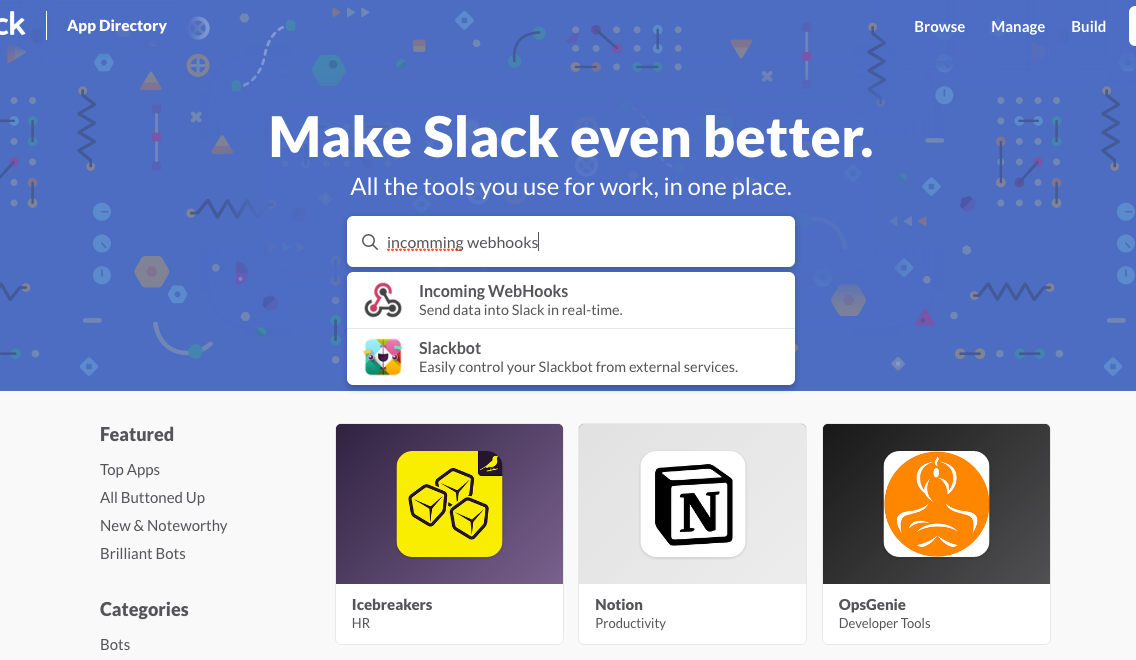
1. 使用 slack 接受消息。
2. 消息能够带有 url， 自动跳转到 prometheus 对应 graph 查询页面。
3. 能自定义颜色。
4. 能够 @ 某人

假设你已注册了 slack 账号，并创建了一个 #test 频道。

一. 为 #test 频道创建一个 incomming webhooks 应用

1. 点击频道标题，选择 Add an app or integration

2. 然后在 app store 中搜索 incomming webhooks，选择第一个

创建成功以后，拷贝 app webhook 地址，以被后面使用。

二. 修改 prometheus rules，添加一些字段

ALERT InstanceStatus

IF up {job="node"}== 0

FOR 10s

LABELS {

instance = "",

}

ANNOTATIONS {

summary = "服务器运行状态",

description = "服务器 已当机超过 20s",

link = "http://localhost:9090/graph#%5B%7B%22range\_input%22%3A%221h%22%2C%22expr%22%3A%22up%7Bjob%3D%5C%22node%5C%22%7D%20%3D%3D%200%22%2C%22tab%22%3A1%7D%5D",

color = "#ff0000",

username = "@sjy"

}

这里，我在 rule 的 ANNOTATIONS 中，添加了 link, color, username 三个字段， 它们分别表示消息外链地址，消息颜色和需要 @ 的人。

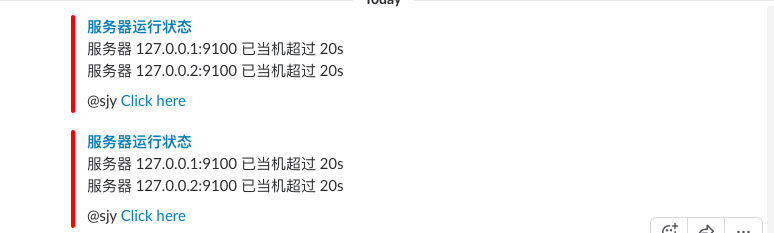
三. 修改 Alertmanager 配置

这里我们将使用到 slack\_configs，配置大致为：

说下配置大致意思：   
1. 按 alertname 分组。   
2. 相同组，如果事件没有恢复，每隔 10s 发送一次（主要为了测试）。   
3. slack\_configs 配置中，使用了 template 语句，通过 CommonAnnotations 查找字段。   
4. 插入外链不仅可以使用 title\_link, 还可以使用 slack link 标记语法 <htttpxxxxxx| Click here>。

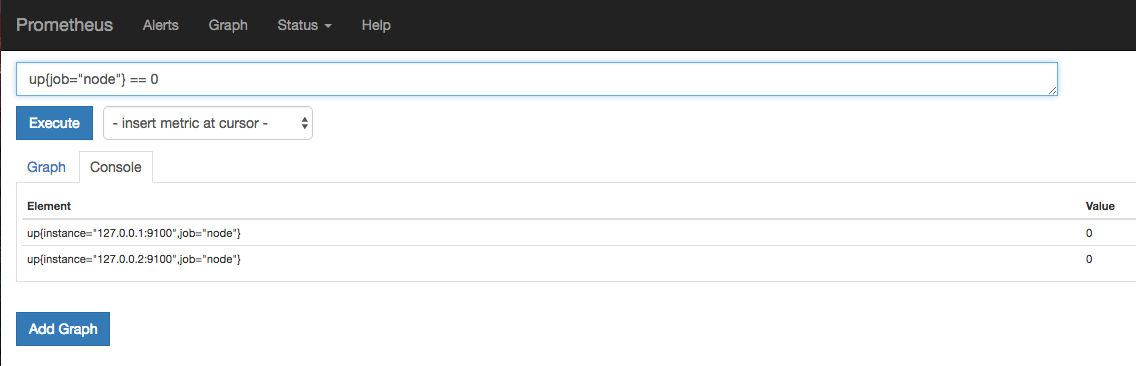
更多 slack 配置，请参考 [incoming-webhooks](https://api.slack.com/incoming-webhooks)。

经过以上配置，我们收到的消息是这样：



消息一条一条的，瞬间清晰很多。有了那几个自定义字段，稍作扩展，你将想到一些有趣的事情，比如自动分配任务，标记不同警报级别。

最后点击 title 或者 Click here， 即可跳转到 Prometheus graph 页面：



真的太方便了，有没有，再也不用担心多个 Prometheus 节点，切换查询的烦恼了。

不得不说，slack 还是非常好用的。经过我测试下来，无论网站，桌面客户端，APP，都没有被墙，消息到达及时，只是网页版，启动较慢。要知道，slack 在 IM 工具里，算很靠谱的了，你不用担心突然关掉之类，我个人比较推荐使用它。

当然如果你还是觉得慢，那么再推荐下零信，号称国内 slack, 他们文档上说是兼容 slack 的。

# How to monitor your system with prometheus

http://www.songjiayang.com/technical/how-to-monitor-your-system-with-prometheus/

14 Sep 2016 in [Technical](http://www.songjiayang.com/categories/index.html#technical)  2 minutes read

**Prometheus 是什么？**

[Prometheus](https://prometheus.io/) 是一个 golang 写的监控告警系统，完全开源，它提供完善的统计模型和友好的查询语句，可以很方便地实现系统机器运行状况监控。

环境准备

* linux amd64 (ubuntu server)
* golang 运行环境 (安装文档 https://golang.org/doc/install)

**Step 1 — 安装 Prometheus Server**

创建下载目录,以便安装过后清理掉

mkdir ~/Download

cd ~/Download

使用 wget 下载 Prometheus 服务（自带数据库）的安装包

wget https://github.com/prometheus/prometheus/releases/download/v1.1.2/prometheus-1.1.2.linux-amd64.tar.gz

创建 Prometheus 目录，用于存放所有 Prometheus 相关的运行服务

mkdir ~/Prometheus

cd ~/Prometheus

使用 tar 解压缩 prometheus-1.1.2.linux-amd64.tar.gz

tar -xvzf ~/Download/prometheus-1.1.2.linux-amd64.tar.gz

cd prometheus-1.1.2.linux-amd64

当解压缩成功后，可以运行 version 检查运行环境是否正常

./prometheus version

如果你看到类似输出，表示你已安装成功:

prometheus, version xxx (branch: master, revision: xxxx)

build user: xxx

build date: xxx

go version: xxx

**Step 2 — 安装 Node Exporter**

Node Exporter 是官方提供的最基本的 Exporter, 提供机器运行相关的信息收集，比如 CPU, Memory, Disk 等等， 它会在 Prometheus Server 定时来拉取数据的时候，提供当前的数据。

使用 wget 下载 Node Exporter

cd ~/Download

wget https://github.com/prometheus/node\_exporter/releases/download/0.12.0/node\_exporter-0.12.0.linux-amd64.tar.gz

使用 tar 解压缩 node\_exporter-0.12.0.linux-amd64.tar.gz

cd ~/Prometheus

tar -xvzf ~/Download/node\_exporter-0.12.0.linux-amd64.tar.gz

cd node\_exporter-0.12.0.linux-amd64

**Step 3 — 启动 Node Exporter**

使用 ./node\_exporter 运行 Node Exporter, 如果看到类似输出，表示运行成功

INFO[0000] Starting node\_exporter (version=0.12.0, branch=master, revision=df8dcd2) source=node\_exporter.go:135

INFO[0000] Build context (go=go1.6.2, user=root@ff68505a5469, date=20160505-22:15:11) source=node\_exporter.go:136

INFO[0000] No directory specified, see --collector.textfile.directory source=textfile.go:57

INFO[0000] Enabled collectors: source=node\_exporter.go:155

INFO[0000] - loadavg source=node\_exporter.go:157

INFO[0000] - textfile source=node\_exporter.go:157

INFO[0000] - time source=node\_exporter.go:157

INFO[0000] Listening on :9100 source=node\_exporter.go:176

当 Node Exporter 运行起来后，可以在浏览器中访问 http://IP:9100/metrics， 将看到类似输出

**# HELP go\_gc\_duration\_seconds A summary of the GC invocation durations.**

**# TYPE go\_gc\_duration\_seconds summary**

go\_gc\_duration\_seconds{quantile="0"} 0

go\_gc\_duration\_seconds{quantile="0.25"} 0

go\_gc\_duration\_seconds{quantile="0.5"} 0

. . .

提示：当然你可以将 node\_exporter 添加到 init 配置里，或者使用 supervisord 作为服务自启动

**Step 4 — 启动 Prometheus Server**

打开 Prometheus 配置文件 prometheus.yml

cd ~/Prometheus/prometheus-1.1.2.linux-amd64

nano prometheus.yml

修改文件，添加如下配置

scrape\_configs:

- job\_name: "node"

scrape\_interval: "15s"

static\_configs:

- targets: ['127.0.0.1:9100']

配置说明：

* scrape\_configs 表示配置一组数据拉取任务相关参数，比如时间间隔，目标地址等
* job\_name 表示一组叫做 node 的任务
* static\_configs 里面的 targets 表示配置的 exporter 地址

详情参考地址 https://prometheus.io/docs/operating/configuration/

保存修改后，启动 Prometheus Server

./prometheus

如果 prometheus 正常启动，你将看到如下信息：

NFO[0000] Starting prometheus (version=1.1.0, branch=master, revision=5ee84a96db6190d4fcdaf4eff74a09b52824a9aa) source=main.go:73

INFO[0000] Build context (go=go1.6.3, user=root@54c6975115bb, date=20160903-19:04:27) source=main.go:74

INFO[0000] Loading configuration file prometheus.yml source=main.go:221

INFO[0000] Loading series map and head chunks... source=storage.go:358

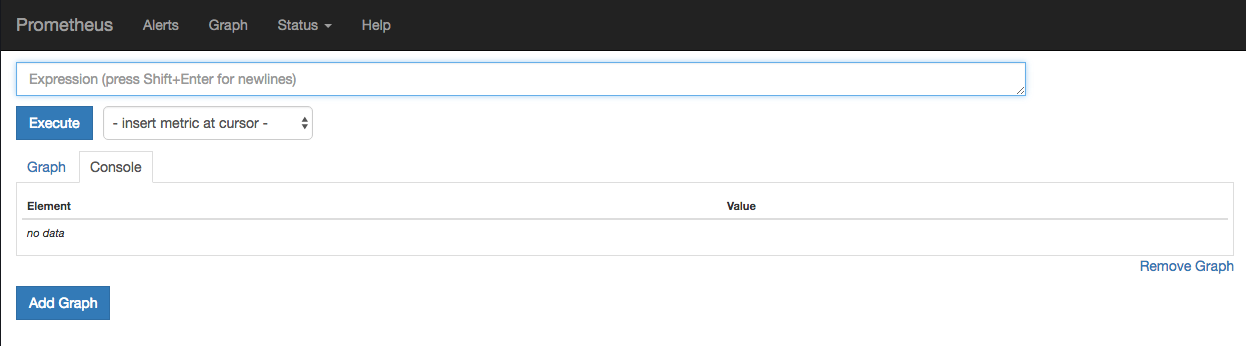
INFO[0000] 974 series loaded. source=storage.go:363

WARN[0000] No AlertManagers configured, not dispatching any alerts source=notifier.go:176

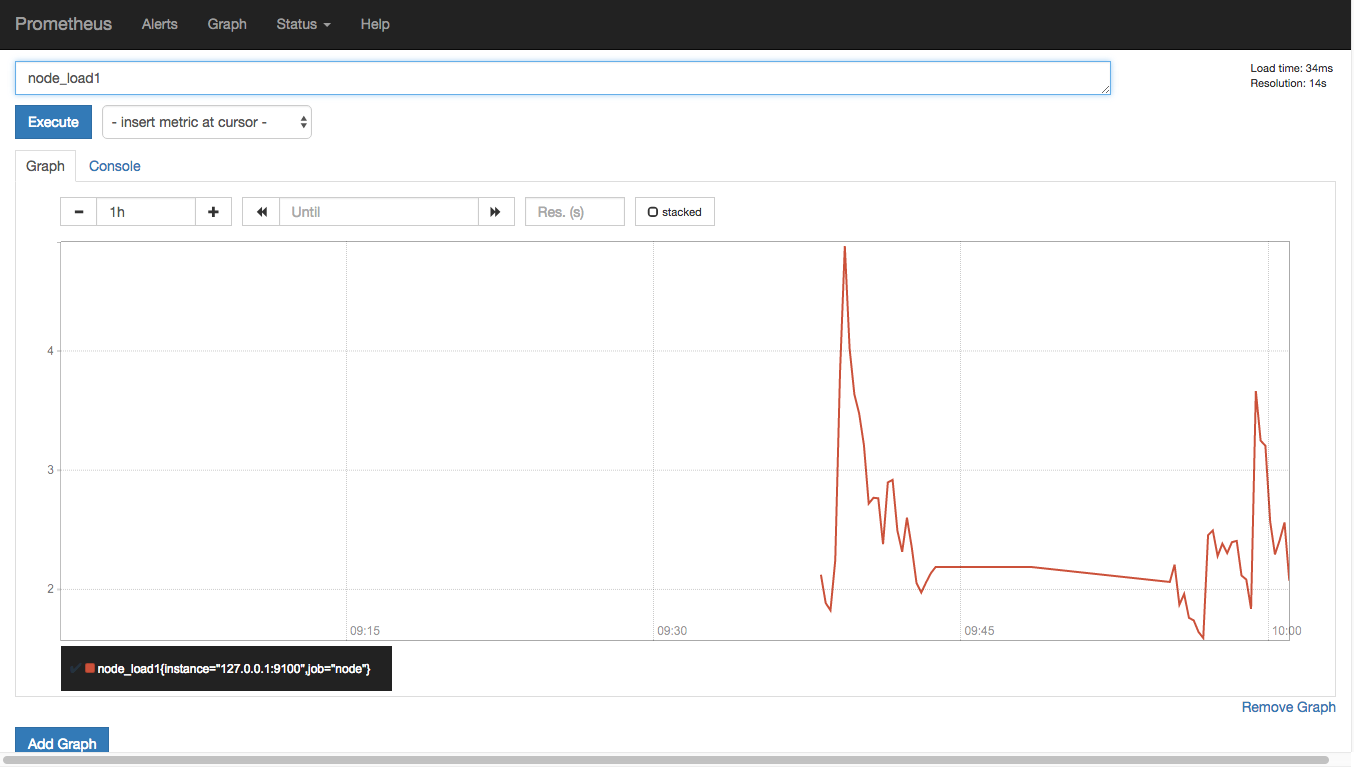
INFO[0000] Starting target manager... source=targetmanager.go:75

INFO[0000] Listening on :9090 source=web.go:233

当 Prometheus 启动后，你可以通过浏览器来访问 http://IP:9090，将看到如下页面



可以使用 PromQL 实现简单查询



提示：你可以将 Prometheus 添加到 init 配置里，或者使用 supervisord 作为服务自启动

**结论**

* 可以看出 Prometheus 安装非常方便，依赖超级少，自带数据库
* 功能蛮强大的，它的 PromQL 很灵活，详情请参考[这里](https://prometheus.io/docs/querying/basics/)
* 自带的 UI 太过于简单，虽然官方提供了 PromDash, 但是已不推荐使用，更好的选择是采用通用的 Dashbord [grafana](http://grafana.org/)
* 如果想要添加更多的机器，可以在 static\_configs 里的 targets 添加对应 IP 地址

# Prometheus with hot reload

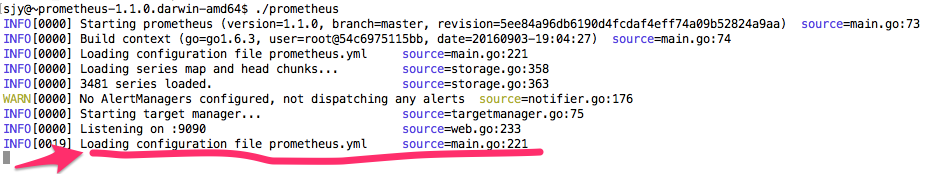
http://www.songjiayang.com/technical/prometheuswith-hot-reload/

当 Prometheus 有配置文件修改，我们想加载新的配置信息而不停掉服务的时候，可以采用 Prometheus 提供的热更新的方法。

热更新的加载方法有两种：

1. kill -HUP pid
2. curl -X POST http://localhost:9090/-/reload

当你采用以上任一方式执行 reload 成功的时候，将在 promtheus log 中看到如下信息：



如果有配置信息填写错误，将导致 reload 失败，你将看到类型如下信息：

ERRO[0161] Error reloading config: couldn't load configuration (-config.file=prometheus.yml): unknown fields in scrape\_config: job\_nae source=main.go:146

提示： 我个人更倾向于采用 curl -X POST 的方式，因为每次 reload 过后， pid 会改变，使用 kill 方式需要找到当前进程号。

再分别说下这两种方式 Prometheus 内部实现：

第一种：通过 kill 命令的 HUP (hang up) 参数实现

首先 Prometheus 在 cmd/promethteus/main.go 中实现了对进程系统调用监听， 如果发现是syscall.SIGHUP 的信号，那么就会执行 reloadConfig 函数。

代码类似:

hup := make(**chan** os.Signal)

signal.Notify(hup, syscall.SIGHUP)

**go** **func**() {

**for** {

**select** {

**case** <-hup:

**if** err := reloadConfig(cfg.configFile, reloadables...); err != **nil** {

log.Errorf("Error reloading config: %s", err)

}

}

}

}()

第二种：通过 web 模块的 /-/reload action 实现。

1. 首先 Prometheus 在 web(web/web.go) 模块中注册了一个 POST 的 action /-/reload, 它的 handler 是 web.reload 函数，该函数主要向 web.reloadCh chan 里面发送一个 chan error。
2. 在 Prometheus 的 cmd/promethteus/main.go 中有个 goroutine 来监听 web 的 reloadCh, 如果有值，那么执行 reloadConfig 函数.

代码类似：

hupReady := make(**chan** bool)

**go** **func**() {

<-hupReady

**for** {

**select** {

**case** rc := <-webHandler.Reload():

**if** err := reloadConfig(cfg.configFile, reloadables...); err != **nil** {

log.Errorf("Error reloading config: %s", err)

rc <- err

} **else** {

rc <- **nil**

}

}

}

}()

总结：Prometheus 内部提供了成熟的 hot reload 方案，这大大方便配置文件的修改和重新加载。

 Tagged with [prometheus 101](http://www.songjiayang.com/tags/index.html#prometheus 101)

## JiaYang Song

http://www.songjiayang.com/me/

**基本档案**

* 姓名: 宋佳洋
* 年龄: 26
* 学历: 大学本科(西南石油大学2013届)
* 英语: 四级
* 邮件: songjiayang1@gmail.com
* 博客：http://songjiayang.github.io
* github: https://github.com/songjiayang

**职业技能**

* 3年互联网行业工作经验，1年带团队经验（4人）；
* 熟悉 \*nux 环境开发和部署;
* 熟悉 Ruby/Rails 开发,有3年使用经验,熟悉常见的gems,有源码阅读习惯。
* 熟悉 MySQL, PostgreSQL, Redis;
* 了解 Python, NodeJs, Java, elixir等其他语言;
* 了解和使用一些前端框架，例如 css (bootstrap,pure)， js (ember, react, backbone) 等。

**工作履历**

* 2009入学西南石油大学,就读计算机科学与技术专业,从此开始了我的程序员之路.
* 2012.10~2013.03 到 团 800 公司做 Ruby 实习.
* 2013.04.25 来上海，工作于现在的公司( GIGA循旅生态科技有限公司)，目前任职技术主管。主要负责 [gigabase](http://www.gigabase.org/en/) 和 [matter](https://giga.matter.build/users/sign_in?locale=zh) 的研发工作。

**岗位职责：**

* 负责公司日常系统功能开发。
* 负责系统发布工作。
* 负责同事代码 Review 和合并工作。

**项目作品：**

* 2011开始接触 Web 开发,自学 JavaEE, 做了一个二手交易平台(inside) ,但是由于种种原因，项目上线不久就停止了.
* 2012 开始接触 Rails,期间开发了个人大学创业项目[微大学](http://vmeal.zuoyouba.com/),一个在线点餐平台。
* 2013 帮助朋友开发了[雪球比特币交易系统](http://snowball.io/)，此系统最终被比特大陆收购。
* 2014 帮助朋友开发了 [Tiny 换汇交易系统](http://tiny-exchange.herokuapp.com/) 。
* 2015 开发了[百善缘绿色有机在线电商平台](http://100.zuoyouba.com/) 。

# 其他

## IT运维利用Slack 传送手机报警讯息-搜狐

http://mt.sohu.com/20161111/n472932125.shtml

【IT168 技术】由于随着个人及企业对信息科技的需求大幅增加，包含移动设备的兴起，社交网站的活跃，以及许多新技术的快速发展，因此在监测设备的使用状况及可以主动发现设备问题并提前预防和故障排除的能力日渐重要，另外，云端服务的兴起，在IT设备资源大量集中化的需求下，管理人员要有足够的能力去分析网络状态及能缩短排除服务异常问题的时间，此仍为当前IT管理人员的一大挑战。 除前述的功能之外，能够快速部署，以及容易操作也会是考虑的要点之一。

　　Slack 是一款团队通讯平台服务，丰富且高度自定义的功能，提供管理团队一个方便、跨平台且多元整合管理的一个沟通管道，且在与第三方串接上的表现令人深刻。而ICINGA 即为一套容易部署以及容易为网管人员短期学习并使用的监控软件，在此本文以Slack和ICINGA 为功能安装，设定及展示之主角，让读者可全面了解并可简单于读者所使用的网络环境下应用ICINGA 。在本篇文章我们将示范如何利用ICINGA 的监控程序并透过Slack的传讯功能发布即时消息至管理团队。

　　主要包括以下内容：

　　1. Slack 注册与安装

　　2. Slack 设定

　　3. ICINGA 设定

　　4. 告警信息传送测试

　　5. 总结

**ICINGA与Slack架构原理图**

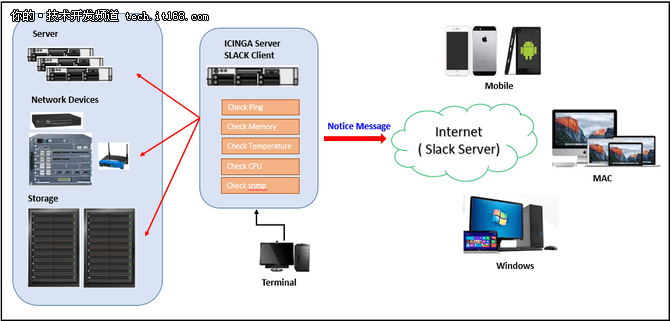
　　Icinga是Nagios 的分支，它提供了全面的监控和报警框架; 是一个容易安装且功能强大的网络监控软件且与 Nagios 一样提供了开放及充足的可扩展性。除可以监控主机之外，任何的设备只要能提供SNMP的服务，如此Icinga即可有充足的信息监控。透过这样的监控协议，我们可以呼叫被监测主机以测量任何可供监控的项目。此外，根据笔者对于其他监控软件的使用经验，如SNSC(IBM System Networking Switch Center)及CNA(Cisco Network Assistant)，两者皆为功能强大的监控软件，但是只针对各自产品做监控而且皆是需付费的软件。因此，相较于Icinga的表现更为全面也较容易符合大众需求。

　　Slack

　　Slack 是「团队沟通平台」, 同时可以在网页版、 Android App、 iPhone App、 Windows 与 Mac 中安装软件使用，跨平台而且实时同步，虽然以网页版的管理功能最完整，但是其它平台也都能满足团队沟通分流的需求, 在 Slack 沟通可以被管理，并转化为有效率的工作流程。在本文中我们把Slack当成是一个client 端，它本身并不提供监控的功能，但透过与Webhook和 ICINGA 的集成，可以提供用户实时报警的功能。

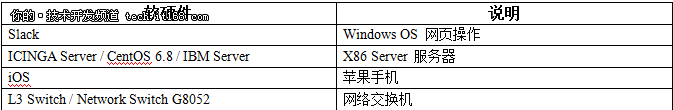
**架构原理图**

　　ICINGA server收集信息的行为如Check\_snmp是经由SNMP的服务来认定每个硬件的ID值来取得数值，回传到server上并以图表纪录，当有报警情况产生，即可透过Slack Clinet 传送讯信息管理员。



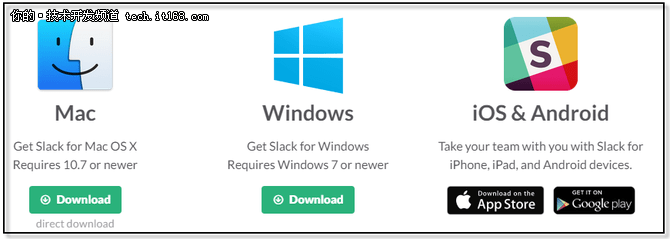
　　▲架构原理图

**操作系统准备**



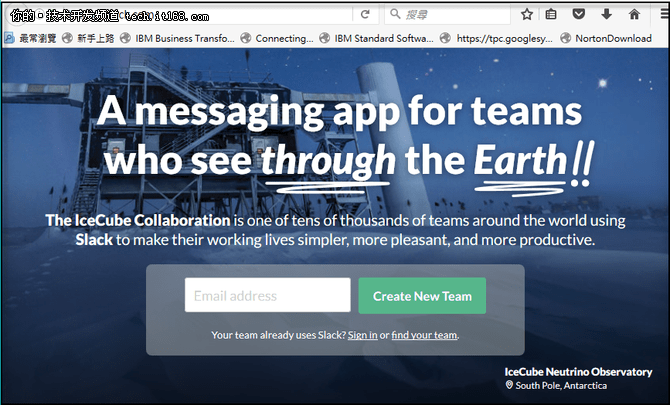
　　1. Slack 网页注册与安装

　　Slack 支持 Mac, Windows 及 iOS & Andorid, 下面将介绍网页注册和手持装置配置过程。

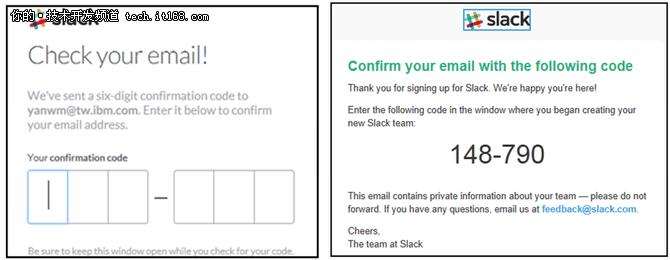


　　1.1 网页注册方式

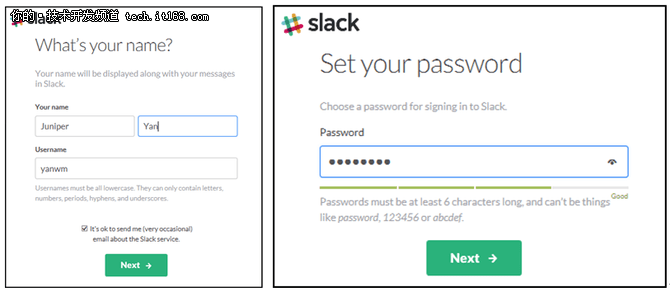
　　利用网页方式新建账号，注册网址为https://slack.com/，输入Email后，选择 " Create New Team"。



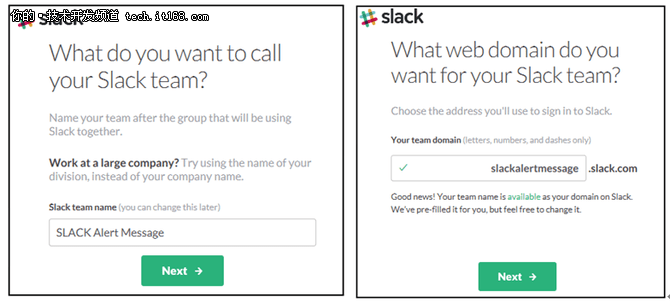
　　电邮会收到由Slack寄来的验证信，在网页上输入验证码。



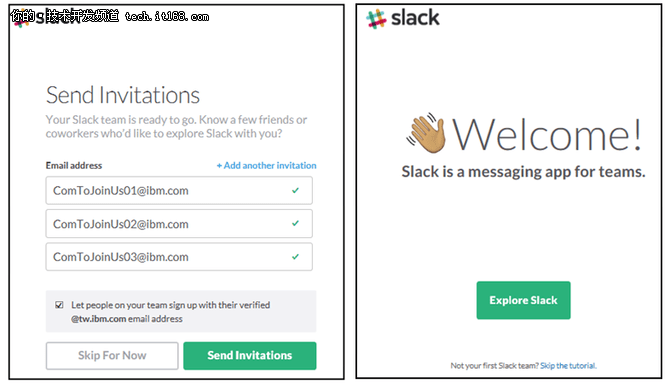
　　设定账号名称与密码，注册完成，进行下一步。



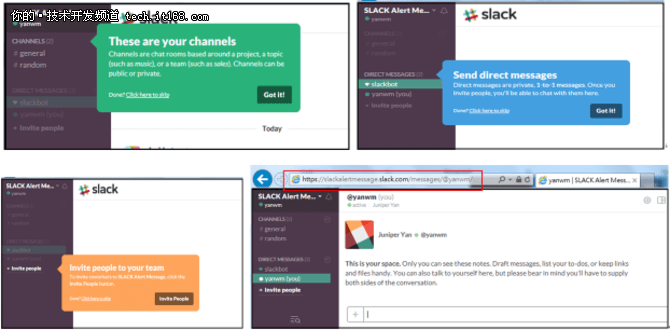
　　接下来设定网页上所要设定的群组名，不另更改的话即为DOMAIN 名。



　　最后可同时寄发邀请信，邀请朋友加入群组。

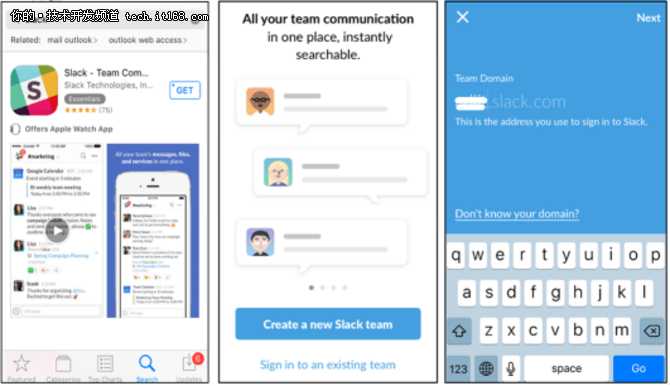


　　可设定邀请人员，并可设定不同群组，让不同群组人员直接讯息连络。

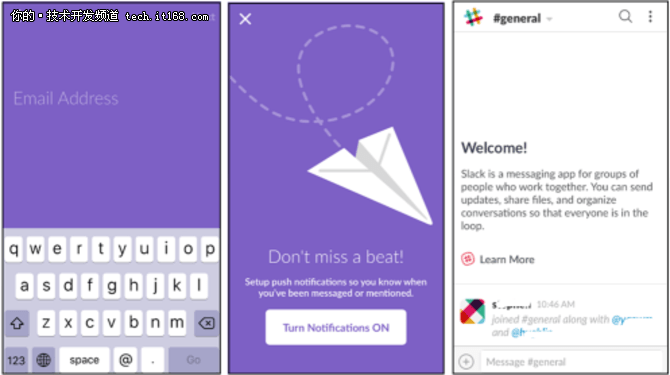


　　1.2 IOS APP 安装与设定

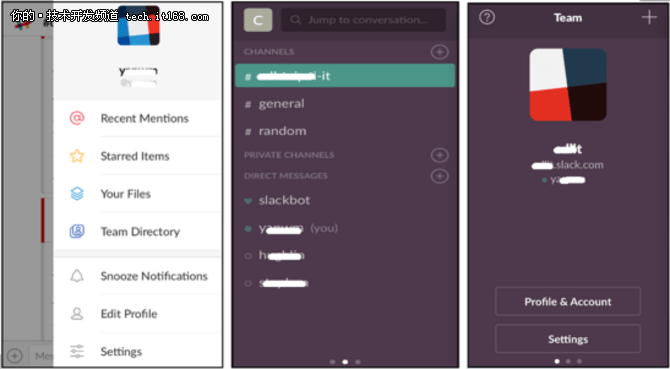
　　Slack 可透过手持装置，来接受实时告警通知，首先我们利用苹果的 Apple Store下载APP，并新建一群组及设定群组名。



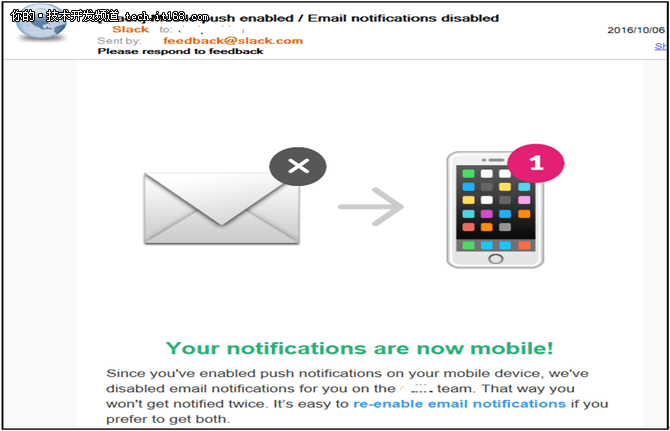
　　输入注册电邮并启用通知功能。



　　登入后，下图为APP主要操控接口。



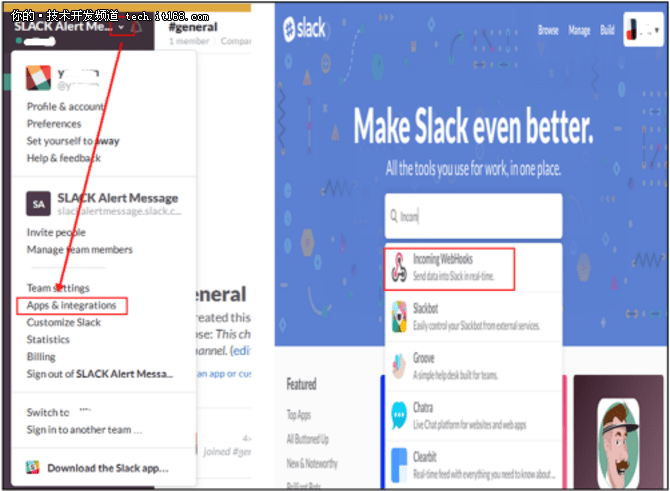
　　当手持装置设定完成后，系统会自动关闭邮件通知功能，改由APP发送简讯通知。



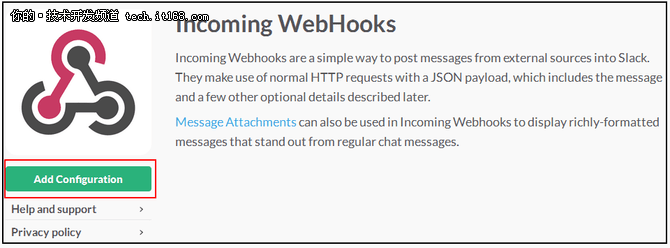
**2. Slack 设定**

　　Slack 提供了 WebHooks，可以实时传送数据，让整个channel 的人都能收到。利用这个特性，设定好事件报警触发的条件后，当管理的硬件出现异常时可通过 WebHooks 向特定的 channel 发送消息，所有在那个channel的管理人员都能立即收到报警消息。

　　首先选择 Apps & Integrations，并安装 “Incoming WebHooks”。



　　选择新增配置。



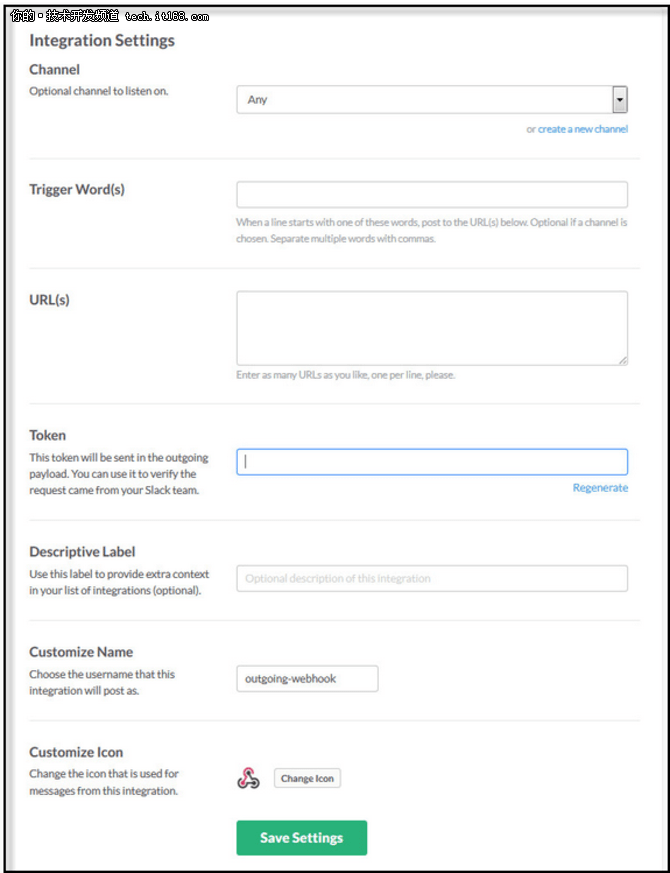
　　可分别设定下列数值:

　　l Channel: 监听的频道。

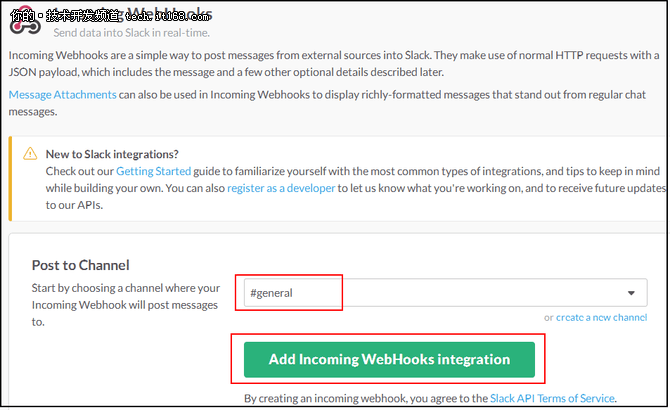
　　l Trigger Word: 触发的关键词，可以逗号分隔。

　　l URL: 接收数据的URL，一行一个。

　　l Token: Slack产生的，可以做为核对身分的依据。



　　下拉选单，并选择要传送讯息的 channel，然后新增 “Incoming WebHooks integration”。

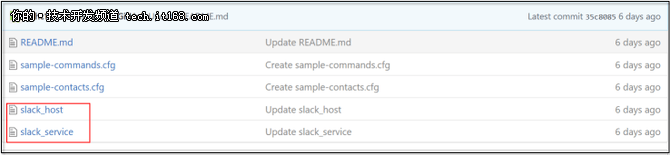


　　记下 Webhook URL，储存后，并复制到ICINGA server上要用的 。

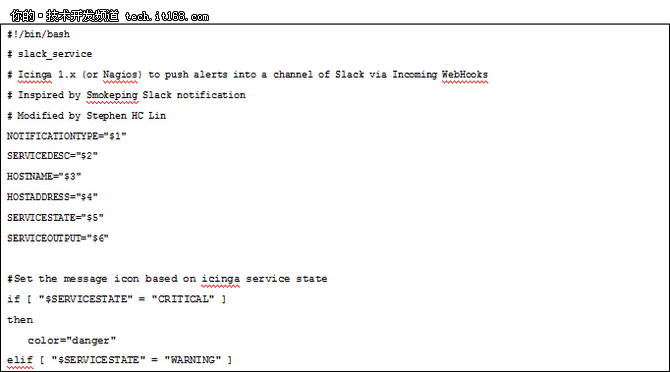


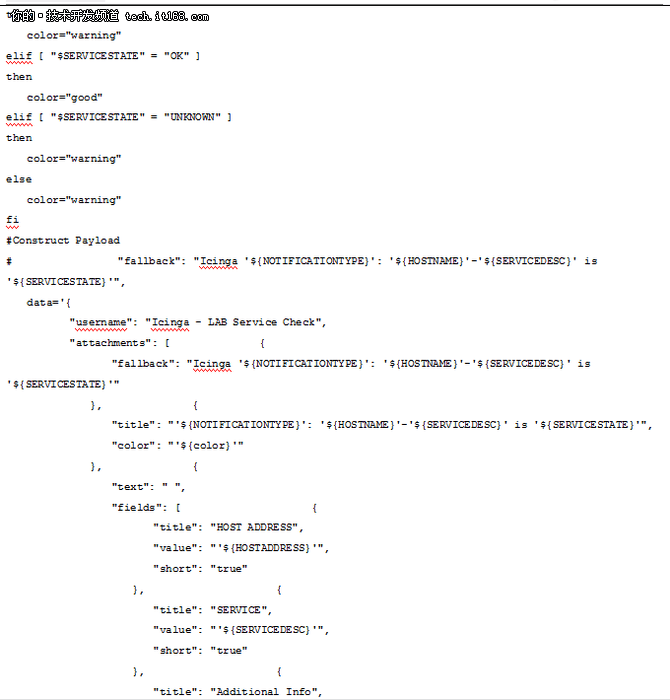
**3. ICINGA 设定**

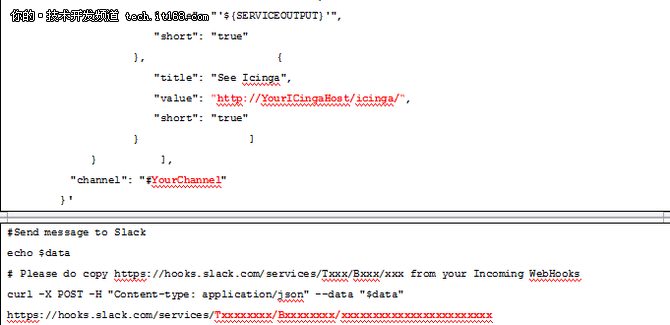
　　3.1. 首先需要建立二个 Slack Notification Shell s，可直接由 Github 上下载 slack\_host和 slack\_service。链接网址为: https://github.com/linhc130/icinga-plugins-slack-notification



　　编辑 slack\_service ，并加上所要连接之Slack 服务的设定参数。





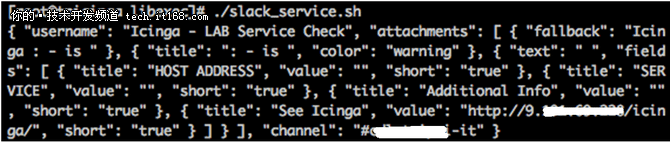


　　编辑 slack\_host.sh，同样的在最后端加入 Slack 的 WebHooks 连接信息。

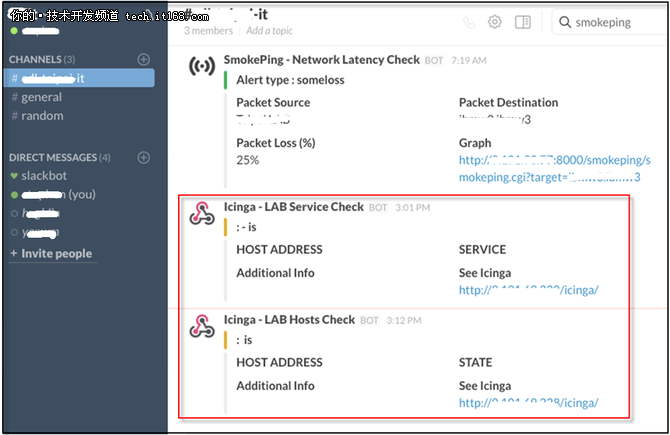


　　3.2. 告警信息Shell s 发送测试

　　执行 slack\_service.sh ，若正常，即会出现一个不带任何信息的告警信息。

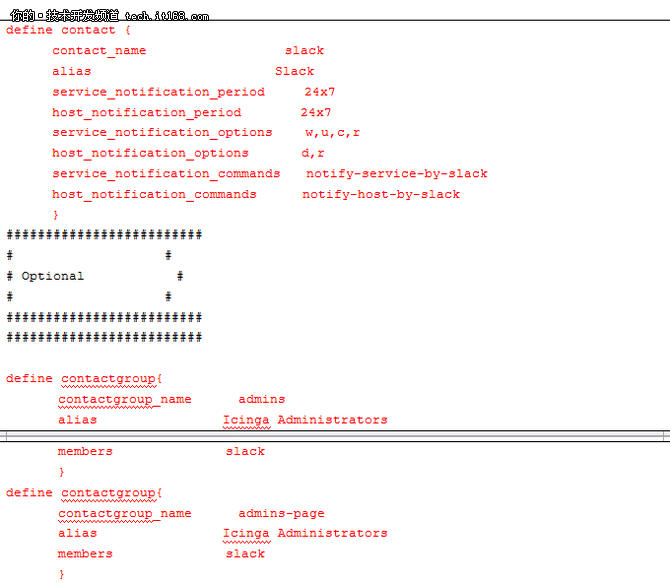


　　ICINGA透过Slack所发出的空白信息。

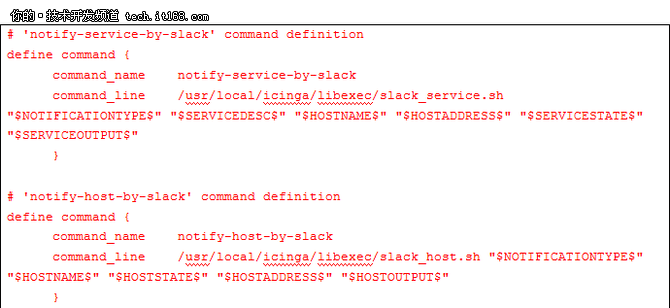


　　3.3. 在 ICINGA设定 Contacts 及 Notification Commands 连接Slack 服务

　　首先在 contacts.cfg 加入下面设定值。

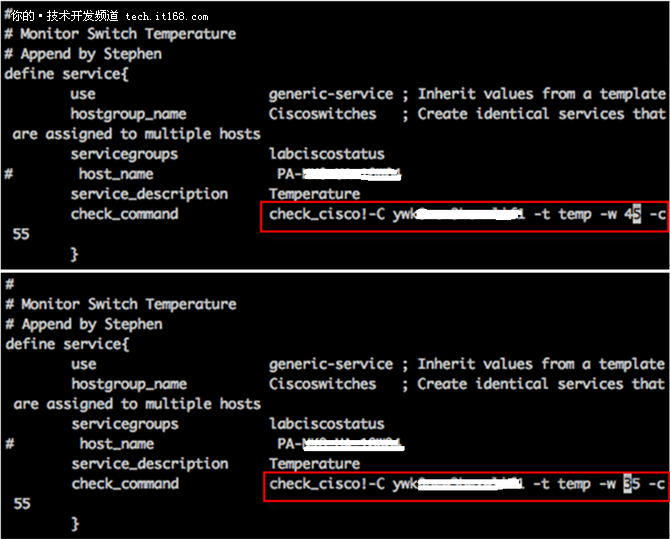


　　并在 commands.cfg 加入下面参数，让系统接收到报警时，去执行 slack\_service.sh 和 slack\_host.sh。

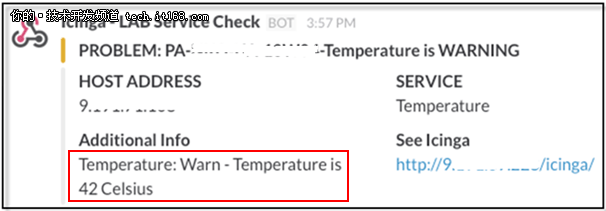


**4. 告警信息传送测试**

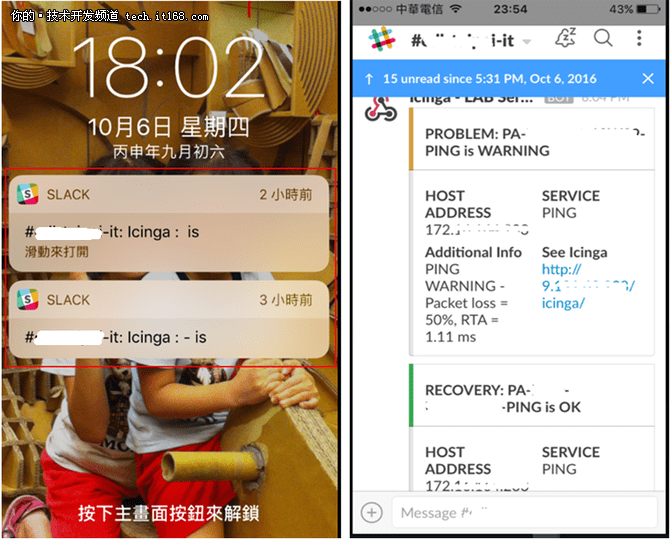
　　变更任一 ICINGA service 报警，如本文例子降低Cisco 温度警报触发度数，使 ICINGA 触发 notification。编辑 switch.cfg 配置文件，在此我们将温度警报设定从 45度降低至 35度 ，重新启动 ICINGA，并查询是否正确收到告警信息。



　　确认可在 Slack 上收到温度告警信息后即可改回,并重启 ICINGA。



　　当事件触发后，查看手机所接收到之信息，开启APP后即可查看告警信息内容。



**5. 总结**

　　维持主机服务的运行，是每个系统管理人员最基本的职责。但受限于人力的考虑，管理人员不可能 24 小时随时监控系统服务的运行。当遇到系统服务发生异常时，能实时通知管理者的监控系统，是每个管理人员所迫切需要的。我们利用 ICINGA 监控软件搭配 Slack 的通讯平台服务，不仅帮助管理人员实时监控系统服务的状态，也能在系统服务发生异常时，立刻以短信通知管理者。让管理人员可以快速处理，减少意外事件的冲击。