

软件工程实训

《面向云智能运维的机器人设计与实现》

姓名：胡睿

班级：电子政务

学号：16340077

日期：2019/06/17

一、阶段任务

《面向云智能运维的机器人设计与实现》项目始于 2019 年 3 月 16 日，止于 2019 年 6 月 18 日，共历时 3 月（合 13 周）。

1、环境搭建

- 1) 服务端：Ubuntu14.04（虚拟主机）
IP: 198.168.188.128
- 2) 客户端：Ubuntu14.04（虚拟主机）
IP: 198.168.188.129

2、工具部署

- 1) hubot+slack hubot adapter 安装配置
在服务端主机安装 hubot，并在 slack 部署 hubot，获取 API Token 后，即可通过终端输入

```
$ HUBOT_SLACK_TOKEN= API Token ./bin/hubot --adapter slack`
```

连接 hubot 和 slack。

- 2) hubot-script-shellcmd 插件
通过 `$ npm install hubot-script-shellcmd`` 安装后，可以使用 hubot 运行 bash 脚本。

- 3) ansible 安装使用

- ① 通过 `$ sudo apt-get update``

```
$ sudo apt-get install ansible`
```

终端命令在服务端主机安装 ansible。

- ② SSH 打通

生成密钥: `$ ssh-keygen -t rsa -P``

传输公钥: `$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.188.129``

- 4) Telegraf + influxdb + grafana 安装使用

- ① 在客户端主机安装 telegraf 监控。
 - ② 在服务端主机安装 influxdb，并创建数据库“telegraf”，设置用户名和密码。
 - ③ 修改 telegraf 配置文件，与 influxdb 连接。

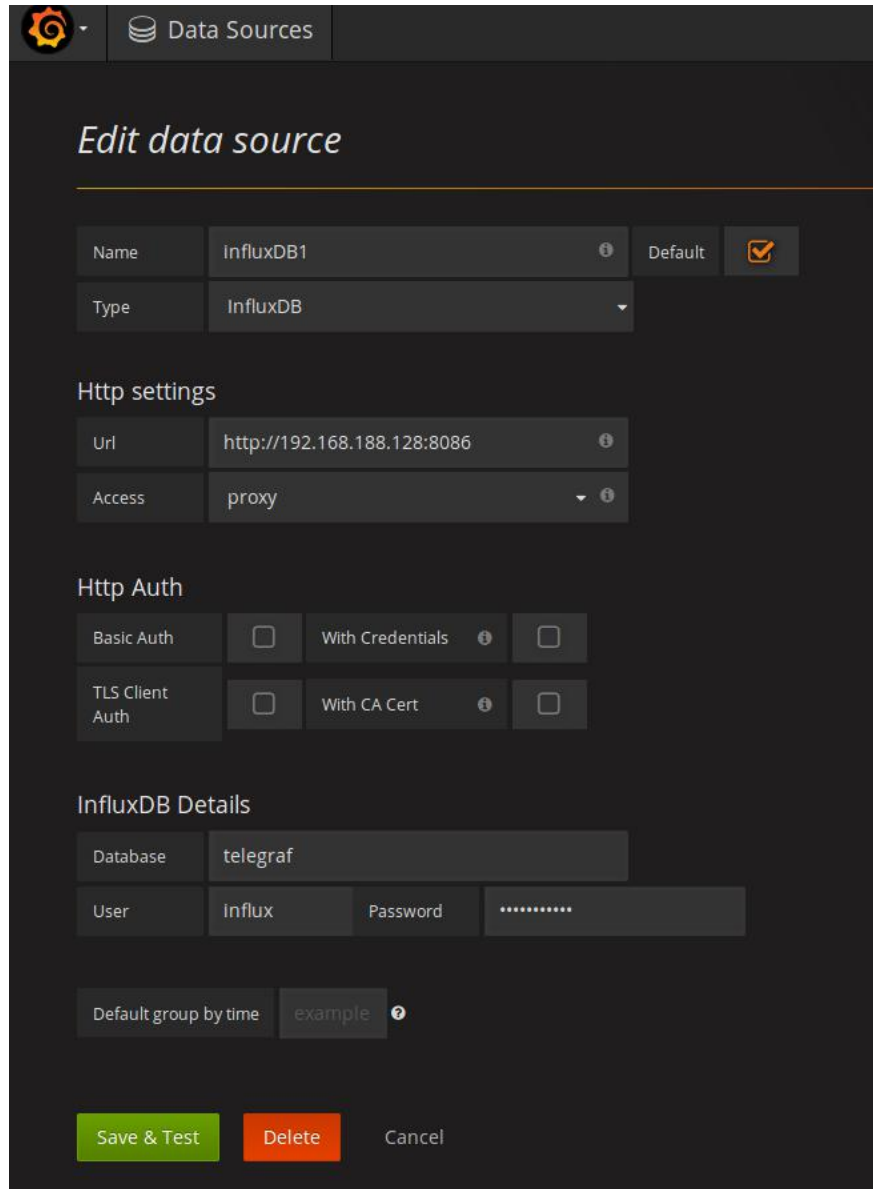
```
# Configuration for influxdb server to send metrics to
[[outputs.influxdb]]
  ## The HTTP or UDP URL for your InfluxDB instance. Each item should be
  ## of the form:
  ##   scheme "://" host [ ":" port]
  ##
  ## Multiple urls can be specified as part of the same cluster,
  ## this means that only ONE of the urls will be written to each interval.
  # urls = ["udp://localhost:8089"] # UDP endpoint example
  urls = ["http://192.168.188.128:8086"] # required
  ## The target database for metrics (telegraf will create it if not exists).
  database = "telegraf" # required

  ## Name of existing retention policy to write to. Empty string writes to
  ## the default retention policy.
  retention_policy = ""
  ## Write consistency (clusters only), can be: "any", "one", "quorum", "all"
  write_consistency = "any"

  ## Write timeout (for the InfluxDB client), formatted as a string.
  ## If not provided, will default to 5s. 0s means no timeout (not recommended).
  timeout = "5s"
  username = "influx"
  password = "influx_pass"
```

④ 在服务端主机安装 Grafana

- a. 在浏览器中输入 `http://192.168.188.128:3000/login`，用户名和密码都是 `admin` 进行登录。创建新的数据源，输入数据库名称、用户名、密码，将 Grafana 与 influxdb 进行连接。



The screenshot shows the 'Edit data source' page in Grafana. The interface is dark-themed. At the top, there's a header with the Grafana logo and a 'Data Sources' tab. Below the header, the title 'Edit data source' is displayed. The main form contains several sections: 1. 'Name' field with 'InfluxDB1' and an 'i' icon, a 'Default' checkbox which is checked, and a 'Type' dropdown set to 'InfluxDB'. 2. 'Http settings' section with a 'Url' field containing 'http://192.168.188.128:8086' and an 'Access' dropdown set to 'proxy'. 3. 'Http Auth' section with two rows: 'Basic Auth' (checkbox unchecked, 'With Credentials' checkbox unchecked) and 'TLS Client Auth' (checkbox unchecked, 'With CA Cert' checkbox unchecked). 4. 'InfluxDB Details' section with a 'Database' field containing 'telegraf', a 'User' field containing 'Influx', and a 'Password' field with masked characters. 5. A 'Default group by time' dropdown set to 'example'. At the bottom, there are three buttons: 'Save & Test' (green), 'Delete' (orange), and 'Cancel' (grey).

Field	Value	Icon
Name	InfluxDB1	i
Type	InfluxDB	▼
Url	http://192.168.188.128:8086	i
Access	proxy	▼ i
Database	telegraf	
User	Influx	
Password	*****	
Default group by time	example	?

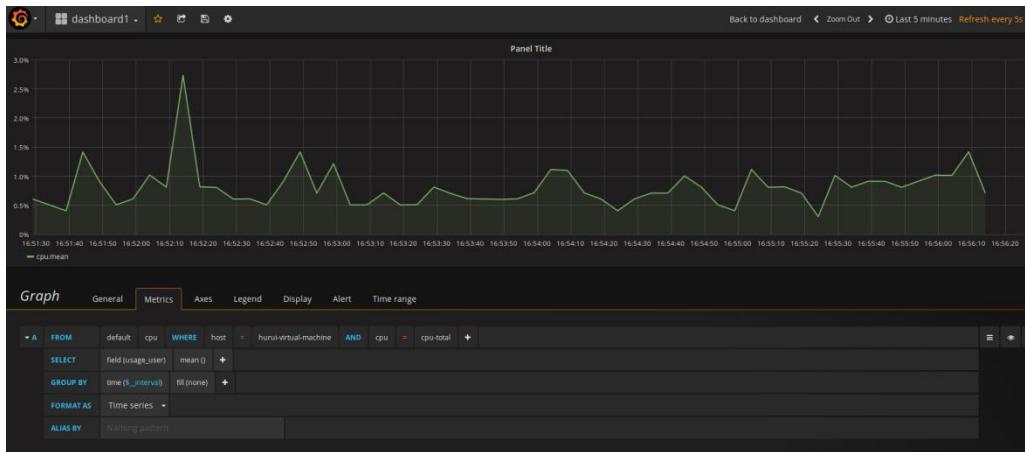
Buttons: Save & Test, Delete, Cancel

- b. 添加仪表盘 Dashboard

- c. 添加面板 Panel

- d. 绘制曲线图

通过 `select` 查询语句绘制 `cpu` 利用率曲线图。



5) Restful API (Flask) 学习使用

3、采集数据

- 1) 在 telegraf 配置文件中设置监控指标，以及监控时采集数据的时间间隔（每 5 秒）将数据写入 influxdb 中。
- 2) 将 influxdb 中的数据导出为 csv 格式文件，以便后续处理。

4、处理数据

在 Flask 程序中读取 csv 格式文件，并进行相应处理。

5、产生异常

- 1) 网络流量异常：在客户端（测试机）上搭建 Apache 本地服务器，在服务端使用 siege 做压力测试，通过在客户端用 tc 命令产生网络时延或丢包。

发现异常点：

```
146,1.5604164567792353e+18,0.6042296290397644,0.503524661064148,351.0,630.0
147,1.5604164567792353e+18,1.3144590854644775,0.7077856659889221,270.0,484.0
148,1.5604164567792353e+18,1.0080645084381104,0.7056451439857483,114.0,178.0
149,1.5604164567792353e+18,1.109989881515503,0.4036327004432678,11.0,4.0
150,1.5604164567792353e+18,0.4032258093357086,0.2016129046678543,7.0,5.0
151,1.5604164567792353e+18,1.1077542304992676,0.6042296290397644,6.0,3.0
152,1.5604164567792353e+18,0.6024096608161926,0.1004016101360321,0.0,0.0
153,1.5604164567792353e+18,0.8048290014266968,0.5030180811882019,5.0,10.0
```

- 2) cpu 利用率异常：在客户端（测试机）上运行包含死循环的程序。

发现异常点：

```
170,1.5604164567792353e+18,0.80402010679245,0.603015065193,763,272.0,486.0
171,1.5604164567792353e+18,2.0263423919677734,0.4052684903,448364,243.0,439.0
172,1.5604164567792353e+18,1.4184397459030151,5.0658559799,9434,256.0,418.0
173,1.5604165942181888e+18,1.4198782444000244,50.6085205078,125,326.0,584.0
174,1.5604165942181888e+18,1.1066398620605469,50.7042236328,125,243.0,431.0
175,1.5604165942181888e+18,1.4127144813537598,6.8617558479,10908,284.0,503.0
176,1.5604165942181888e+18,1.6096580028533936,0.7042253613,471985,248.0,448.0
```

6、算法选取

- 1) 试用密度估计算法（未采用）
- 2) 选用孤立森林算法 IsolationForest

综合考虑孤立森林算法特性：

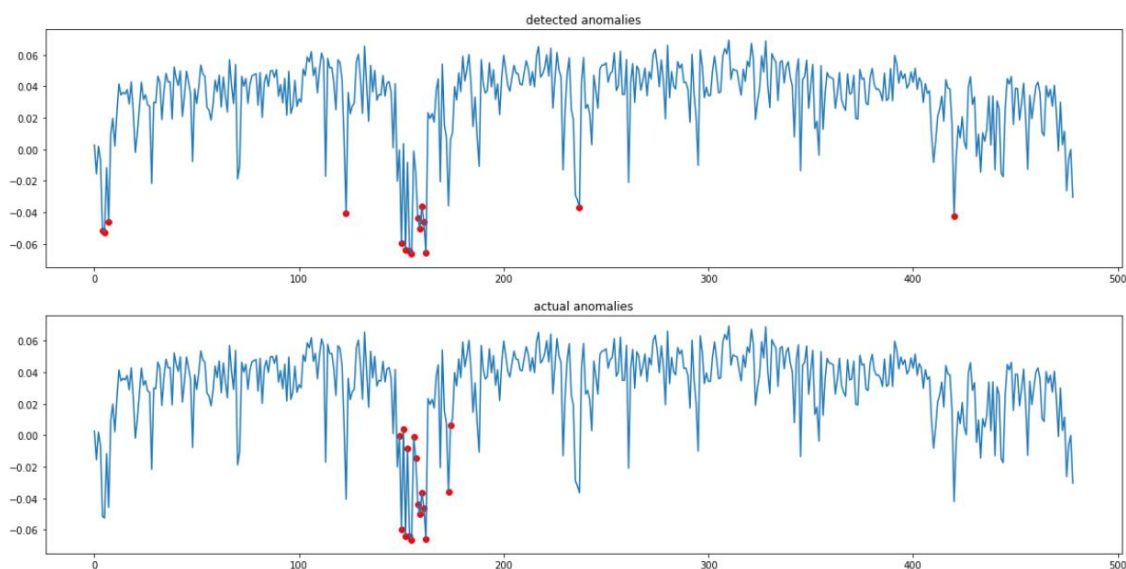
- ① 可以找出全局数据中的异常点；
 - ② 可以处理高维数据。
- 3) 试用指数平滑算法（未采用）

7、结果分析

调整孤立森林算法传递参数值，比较检测结果与实际异常点，计算误报点数、漏报点数以及准确率。

`IsolationForest(n_estimators=200, max_features=1, max_samples=481, contamination=0.03, random_state=None)`

- 1) 下图中，第一个图为算法检测到的异常点，第二个图为实际的异常点。横坐标为数据点索引值，纵坐标为异常程度，数值越小表明异常程度越高。



误差分析：调用函数参数选取的异常点比例为 0.03，即：若异常程度从小到大排列，前 3%的点将被归为异常点。由于数据包含 4 个指标（有一个无异常指标），算法会“公平地”考虑每一维数据的异常程度，如果无异常指标中出现少许数值波动点，很有可能会被归为异常点。

2) **样本数：**481

异常点数：16

实际异常点比例：0.033

准确率 = ((异常点数-漏报点数)/异常点数-误报点数/(样本数-异常点数))×100%

(说明：由于未能在网上找到可靠的孤立森林算法准确率计算公式，上为结合误报点与漏报点的粗略计算公式。)

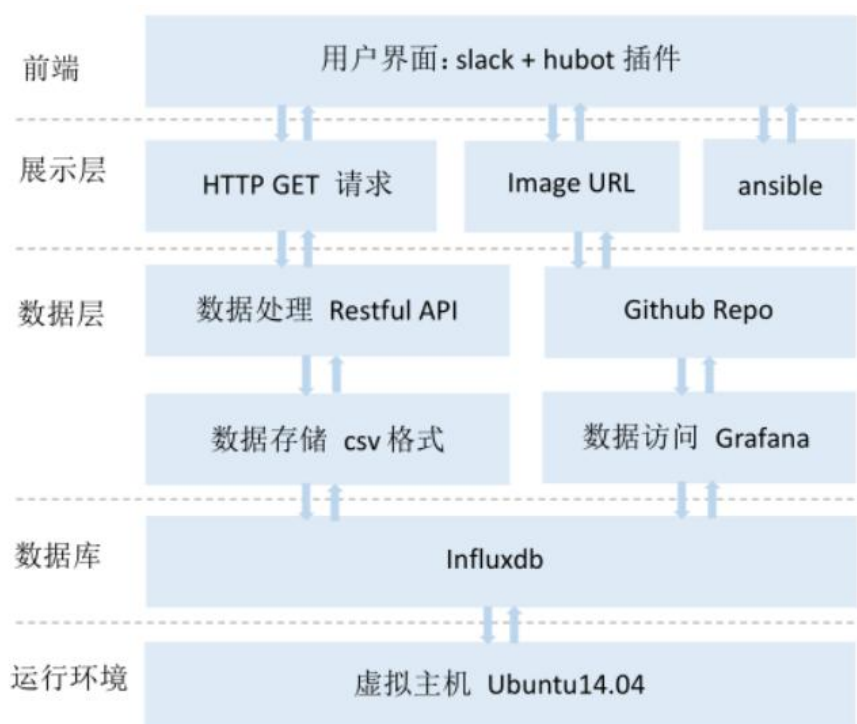
下表选取了两个变量分别为：森林中树的棵数以及设定的异常点比例。

树的数目	异常点比例	误报点数	漏报点数	准确率
100	0.02	7	13	17.25%
	0.03	9	10	35.56%
	0.05	10	2	85.35%
300	0.02	7	13	17.25%
	0.03	9	10	35.56%
	0.05	9	1	91.81%
500	0.02	7	13	17.25%
	0.03	10	11	29.10%
	0.05	10	2	85.35%

分析：森林中树的数目增加，准确率不会有太大变化（采用 100-300 即可）
异常点比例设置略高于实际异常点比例，准确率会相应提升，缺陷是误报点数也会随之提升。

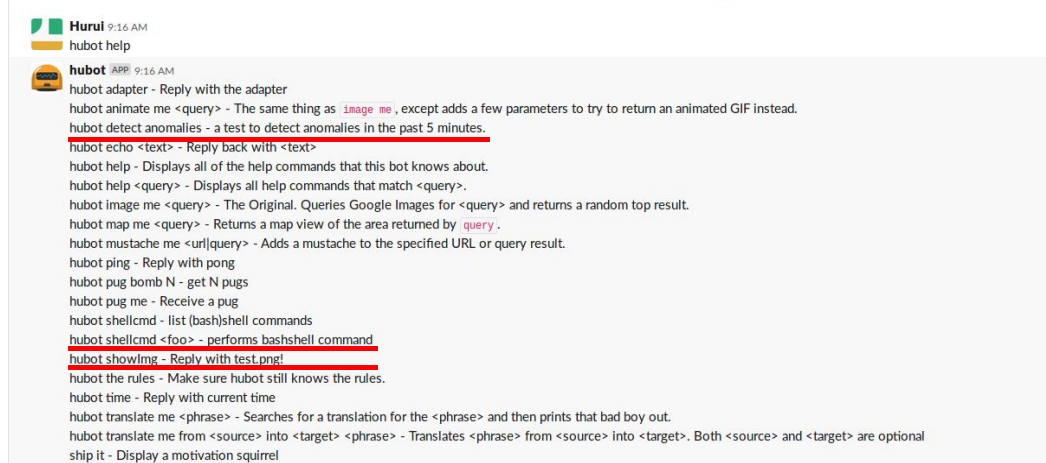
二、完成情况

系统架构



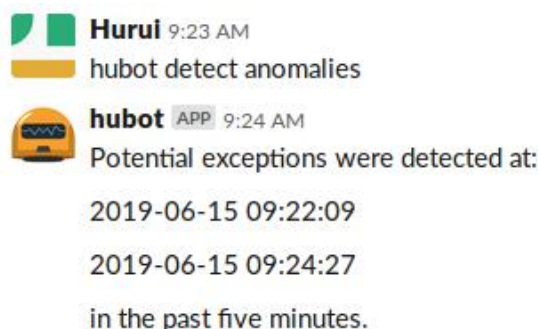
下图显示了我编写的自定义脚本运行命令。

> hubot help



1、检测 5 分钟内异常

> hubot detect anomalies



完成模块及流程：

1) hubot+slack

- ① 用户在 slack 的 hubot 会话窗口输入命令`hubot detect anomalies`。
- ② 等待一段时间，hubot 返回最近五分钟异常产生时间。

2) 自定义脚本 coffeescript

- ① 通过 robot.respond 监听对 hubot 说的「输入」。
- ② 通过 child_process.exec 运行 bash 文件。
- ③ 通过 HTTP GET 方法请求服务器中数据（异常产生时间），返回给用户。

3) bash

将 influxdb 中最近 5 分钟的数据（CPU 利用率、网卡收发数据包数）写入 csv 格式文件中。

4) Rest API

- ① Flask 读取 csv 文件中的数据
- ② 处理数据

processData(data)

a. 参数

data: 转换成数组格式的原始数据

包括时间戳、2 个 cpu 指标:系统 cpu 利用率 usage_system、用户 cpu 利用率 usage_user

以及 2 个 net 指标: 网络数据包收发数 packets_recv、packets_send

b. 返回值:

d2: 处理后的数据, 元素类型为浮点数

result: 处理后的数据, 元素类型为字符串

③ 通过孤立森林的算法检测异常

```
ilf = IsolationForest(n_estimators=300, max_features=2, max_samples=481,
                      contamination=0.03, random_state=None)
```

④ 时间格式转换

generate_time(timestamp)

a. 参数:

timestamp: 时间戳 list

b. 返回值:

otherStyleTime: 年/月/日 时/分/秒 list

2、使用 ansible 命令查看两个主机系统 cpu 使用情况

> hubot shellcmd cpu

```
Hurui 9:17 AM
hubot shellcmd cpu

hubot APP 9:17 AM
192.168.188.128 | CHANGED | rc=0 >> top - 09:17:42 up 7 min, 5 users, load average: 0.51, 0.63, 0.35
Tasks: 268 total, 1 running, 267 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.0 us, 9.1 sy, 0.2 ni, 71.5 id, 13.0 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem: 4028204 total, 1841716 used, 2186488 free, 126428 buffers
KiB Swap: 2094076 total, 0 used, 2094076 free. 681540 cached Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3398 raven   20  0 29184 3332 2816 R 15.5 0.1   0:00.06 top -bn 1+
 3292 raven   20  0 202760 46080 9460 S 10.3 1.1   0:03.68 /usr/bin/+
 1546 grafana 20  0 294724 25640 16484 S 5.2 0.6   0:01.75 /usr/sbin+
 2980 raven   20  0 910844 51580 27540 S 5.2 1.3   0:06.24 node node+

192.168.188.129 | CHANGED | rc=0 >> top - 09:17:43 up 7 min, 2 users, load average: 1.15, 1.08, 0.55
Tasks: 254 total, 1 running, 253 sleeping, 0 stopped, 0 zombie
%Cpu(s): 10.9 us, 15.4 sy, 0.3 ni, 63.4 id, 9.7 wa, 0.0 hi, 0.3 si, 0.0 st
KiB Mem: 4028204 total, 1908924 used, 2119280 free, 76372 buffers
KiB Swap: 2094076 total, 0 used, 2094076 free. 653972 cached Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3043 hurui   20  0 1832140 353316 113492 S 12.9 8.8   1:08.60 /usr/lib/+
 3721 hurui   20  0 29184 3356 2844 R 4.3 0.1   0:00.08 top -bn 1+
```

实现过程:

1) 在 /etc/ansible/hosts 文件下添加分组模块如下:

```
[test]
```

```
192.168.188.128
```

```
192.168.188.129
```

```
[dev]
```

```
192.168.188.129
```

```
ansible_connection=ssh
```



```
ansible_ssh_user=hurui
ansible_ssh_port=22
```

- 2) 编写 shell 脚本，修改权限为可执行程序，并将文件拷贝到 hubot 安装目录的 bash/handlers 文件夹下。

```
#!/bin/bash
```

```
ansible test -m shell -a 'top -bn 1 -i -c'
```

- 3) 在与 hubot 的会话中输入`hubot shellcmd cpu`命令即可得到两台主机的 cpu 使用情况结果。

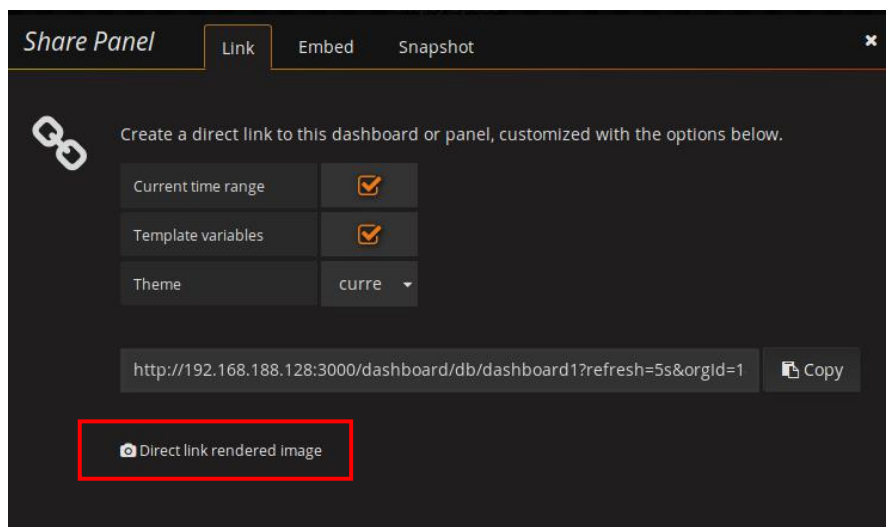
3、显示前 5 分钟 Grafana Panel 中 cpu 使用情况图像

> hubot showImg



实现过程：

- 1) 在 Grafana 的 Dashboard 中点击 panel 选择“share”，选择“Direct link rendered image”。



- 2) 获取图片链接，通过 shell 脚本上传到 Github 的项目仓库中。
- 3) 通过 coffeescript 脚本将 Github 中的图片链接返回给用户。

三、 遇到的问题及解决方法

1、选取算法的两次尝试 (2 周)

- 1) 基于密度估计的算法
选择原因：实现简单。
未使用原因：适用于单维或者低维。
- 2) 指数平滑算法
选择原因：可以检测出局部异常。
未使用原因：时间不够，未能实现。

2、Grafana 返回图片到 hubot

1) 问题一：纯手动操作

流程：导出图片 -> curl 下载图片 -> push 到 github -> 编写 scripts 返回图片

解决办法：将命令行的操作写入脚本文件 -> scripts 中运行

重难点：

- ① 字符串拼接：为了使 hubot 返回最近时间内的 Grafana 图像，需要在命令行重新拼接图片链接 url。
- ② Github 免密 push 操作
使用 `$ git remote set-url origin git@github.com:USERNAME/REPOSITORY.git`
命令将 remote's url 改为 SSH。
- ③ 等 shell 脚本执行完，图片完成传输后，再返回链接。

2) 问题二：无法预览图片

① 图片在本地服务器

原因：slack 无法解析本地 ip 地址。

解决办法：将图片传到公网上（如：Github）

② 同一链接无法预览多次

原因：在同一个会话中，如果 hubot 在最近一段时间返回过相同的图片链接，则不会生成预览图片。

解决办法：生成随机数参数传入 shell 脚本文件，并作为图片文件名，以避免每次生成链接相同。

3) 问题三：异常检测 hubot 返回结果不准确（未解决）

使用 hubot detect anomalies 命令时，无论是否真的有“异常”存在，至少有 1-2 个检测到的异常点。

原因：使用孤立森林算法检测异常，会根据传入参数“contamination”的值来确定数据中异常点比例。

设想解决办法：修改孤立森林算法，通过异常程度判断有无异常。

四、 后续工作

1、改进算法

由于被检测数据存储于时序数据库中, 我们需要通过数据时序性这一特点进一步检测局部异常, 提高算法检测的准确度。

2、日志分析

通过 Filebeat 采集服务日志内容、容器日志内容以及系统日志内容, 使用 Elasticsearch 短期存储或 HDFS 长期存储工具存储日志数据, 通过智能分析算法进行分析。

3、根因定位

结合系统指标以及日志数据分析结果进行根因定位。

4、整合部署

待项目功能完善后可制作成易于使用的软件。

Github 项目仓库地址: <https://github.com/170226/IOM/tree/master>