서비스 운용 시각화 및 IoT 데이터 분석 UI

Document No. 1
Version 1.0
Date 2017-11-29
Author(s) 최지호, 김주민, 유소현, 송영진

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2017. 09. 20	최지호,김주민,유소현,송영진	
0.2	2017. 10. 21	최지호,김주민,유소현,송영진	
0.3	2017. 11. 01	최지호,김주민,유소현,송영진	
중간 - 0.4	2017. 11. 07	최지호,김주민,유소현,송영진	
최종	2017. 11. 29	최지호,김주민,유소현,송영진	

본 연구는 한국정보화진흥원(NIA)의 미래네트워크선도시험망 (KOREN) 사업 지원과제의 연구결과로 수행되었음 (17-951-00-001).
This research was one of KOREN projects supported by National Information Society Agency (17-951-00-001).

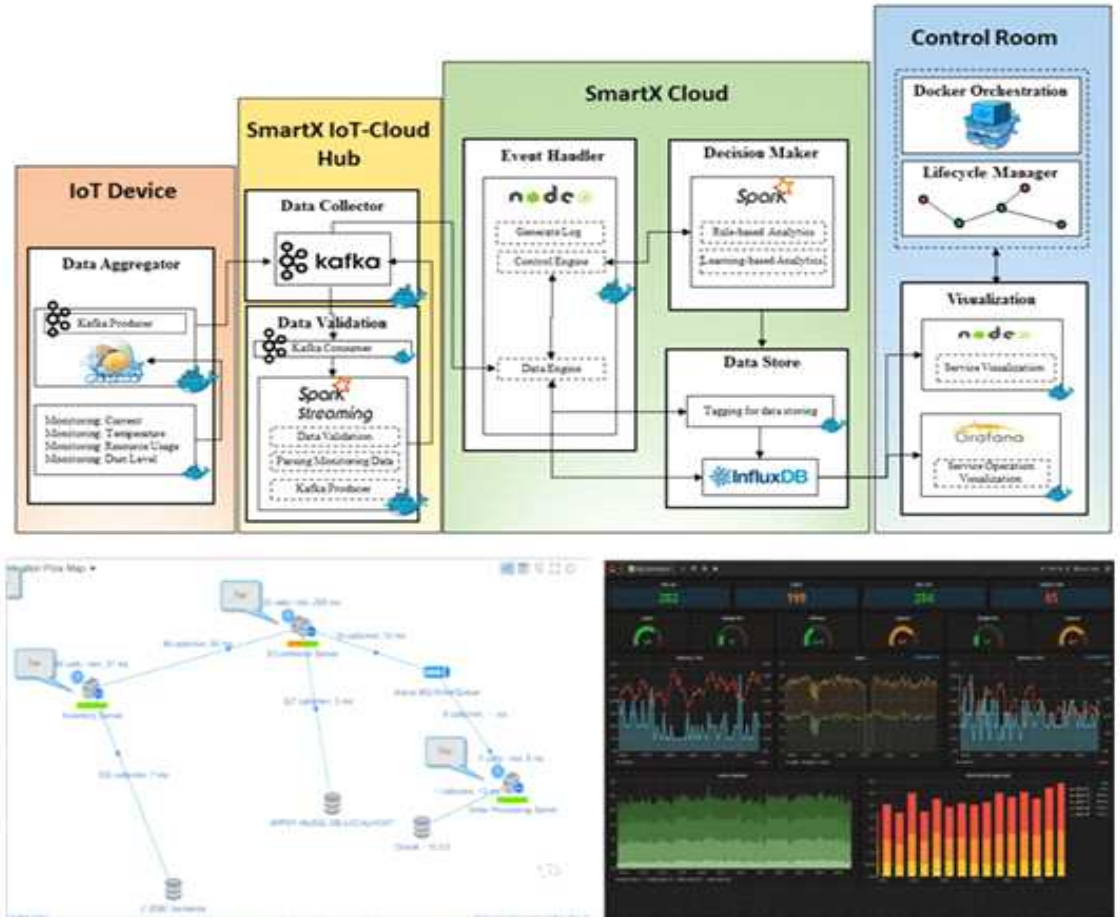
그림 목차

Contents

#8. 서비스 운용 시각화 및 IoT 데이터 분석

#9. 서비스 운용 시각화 및 IoT 데이터 분석 UI

1. 개요



[그림-1] 서비스 운용 시각화 및 IoT 데이터 분석 UI

o IoT-Cloud 서비스의 원활한 운용 및 데이터 분석을 위하여 서비스 모듈의 연동을 통한 서비스 운용을 일목요연하게 파악할 수 있는 “서비스 운용 시각화 UI”와 수집된 IoT 데이터를 다양한 측면에서 분석할 수 있는 “IoT 데이터 분석 UI”를 개발한다. 서비스 운용 시각화 UI는 현재 운용되는 서비스의 모듈에서의 데이터 흐름과 각 모듈의 현황을 파악할 수 있으며, 모듈의 상태 이력 등에 대한 정보를 파악할 수 있어, 서비스 운용을 효과적으로 지원하는 도구로서 활용된다. IoT 데이터 분석 UI의 경우는 실시간 IoT 데이터를 분리해서 분석하여, 서비스의 운용에서의 문제점 파악은 물론 IoT 데이터의 기술 통계 분석 및 다양한 형태의 차트를 통한 데이터 분석을 제공하는 도구로서 활용된다.

8.2 Softwares

8.2.1 Grafana

[그림-2] 예시1) Grafana Dashboard

[그림-3] 예시2) Grafana Dashboard

Grafana는 시각화 도구의 한 종류이다. 쌓여 있는 데이터를 동적으로 시각화 할 수 있도록 해주는 오픈소스이다.

8.2.2 InfluxDB

o Time-series DB: 시계열 데이터를 저장하고 활용하는데에 특화된 DataBase

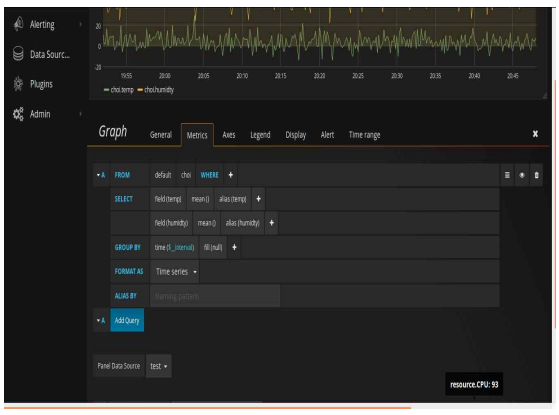
o 전체적으로 Go 로 작성. 외부 종속성없이 단일 바이너리로 컴파일

o Graphite ,collected ,OpenTSDB와같은 다양한 플러그인 제공

o 간단하고 고성능의 HTTP(S) API 작성 및 쿼리

o 웹 관리 인터페이스가 내장

o 태크를 사용하여 빠르고 효율적인 쿼리를 위해 시리즈를 인덱싱 함



[그림-4] 예시1) Grafana에서 Query문 작성

[그림-5] 예시2) Grafana InfluxDB연동

8.3 설치 및 환경설정

8.3.1 설치 개요 및 사전 준비

8.3.2 Docker 설치 및 환경구성

o Docker 설치

먼저 Docker를 설치하기 위해 터미널에 다음과 같은 명령어를 입력한다.
curl -s https://get.docker.com/ | sudo sh
curl이 없다면 먼저 sudo apt-get install curl 명령어로 설치해준다.

Docker 버전을 확인하여 설치가 정상적으로 되었는지 확인한다.
sudo docker version

```
[sudo] password for jumin:
Client:
 Version:      17.07.0-ce
 API version:  1.31
 Go version:   go1.8.3
 Git commit:   8784753
 Built:        Tue Aug 29 17:42:53 2017
 OS/Arch:      linux/amd64

Server:
 Version:      17.07.0-ce
 API version:  1.31 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   8784753
 Built:        Tue Aug 29 17:41:43 2017
 OS/Arch:      linux/amd64
 Experimental: false
```

[그림-6] 도커 버전 확인

Server에 내용이 올바르게 표시되지 않는다면 잘못 설치된 것이다.

※도커를 실행하기 위한 kernel 버전은 3.10.x 이상이므로 ubuntu 14.04 이상을 사용하면 큰 문제가 없으나 kernel의 버전이 낮을 경우 제대로 동작을 안 하거나 문제가 생길 수 있다.
※docker는 기본적으로 관리자 권한이 필요하니 docker 명령어에는 root 계정이 아닌 경우 꼭 sudo를 붙여준다.

o 자주 사용하는 옵션

도커 명령어 사용방법

docker run <옵션> <이미지 이름,ID> <명령> <매개 변수>

run - 컨테이너를 생성하고 시작하는 명령어
images - 이미지 목록 조회
ps - 현재 실행중인 컨테이너 목록을 보여주는 옵션
ps -a -생성한 전체 컨테이너 목록을 보여주는 옵션
exec -해당 컨테이너로 접속 하는 옵션
c,m,cpuset - 컨테이너 자원 할당에 관한 옵션
i,- 입력에대한 출력을 나타내는말 즉, interactive한 흐름을 만들어 주는 옵션
volume -도커의 디렉토리 공유를 위한 옵션
p -외부와 연결하는 옵션 ,포트를 외부와 도커와 연결하는 옵션

o . 환경구성

Docker를 사용하기 위해서는 우선 docker service를 시작해야 한다.
터미널에 'sudo service docker start'를 입력한다.

이제 환경을 구성할 Ubuntu를 설치한다.
search 명령으로 원하는 이미지를 검색할 수 있다.
Ubuntu 이미지를 사용할 것 이므로 Ubuntu를 검색해본다.
sudo docker search ubuntu

```
NAME                                                     DESCRIPTION
                        STARS     OFFICIAL   AUTOMATED
ubuntu                                                   Ubuntu is a Debian-base
d Linux operating s...  6560      [OK]
```

[그림-7] Ubuntu Image검색

다양한 버전의 Ubuntu OS가 표시되지만 Debian-base의 최상단 이미지를 사용하다.

pull명령으로 이미지의 이름을 입력하면 해당 이미지를 받을 수 있다.
별다른 옵션이 없으면 자동으로 가장 최근 버전으로 받아진다.
sudo docker pull ubuntu

```
Using default tag: latest

latest: Pulling from library/ubuntu
d5c6f90da05d: Pull complete
bbbe761fcb56: Pull complete
7afa5ede606f: Pull complete
f6b7253b56f4: Pull complete
2b8db33536d4: Pull complete
Digest: sha256:2b9285d3e340ae9d4297f83fed6a9563493945935fc787e98cc32a69f568764
1
Status: Downloaded newer image for ubuntu:latest
```

[그림-8] 도커 Image 다운로드

run 명령을 사용하여 받은 Image를 Container에 올린다.

sudo docker run -i -t --net=host -e DISPLAY -v /tmp/.X11-unix –-name ubuntu ubuntu /bin/bash
-it : 명령은 터미널 입력을 위한 옵션
--net=host : Host OS의 네트워크를 사용하는 옵션
-e DISPLAY -v /tmp/.X11-unix : 터미널에서 GUI를 사용하기 위한 옵션
--name : Container에 자유롭게 이름을 부여한다.
/bin/bash : Ubuntu Container 안의 bash 쉘을 사용한다.

[그림-9] 컨테이너 안으로 들어간 모습

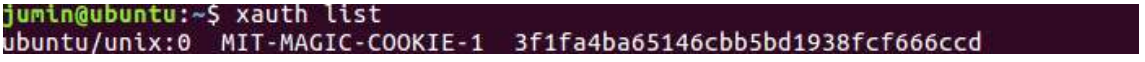
새로 만든 Ubuntu Container는 버전이 낮으므로 update를 해준다.
apt-get update
※Ubuntu Container는 루트 계정이 default이므로 명령에 sudo를 붙일 필요가 없다.

Grafana와 Wordpress는 GUI가 필요한 웹기반 Software이기 때문에 Firefox, Chrome 등 웹 브라우저의 설치가 필요하다. 비교적 설치가 간편한 firefox를 설치한다.
apt-get install firefox

Docker Container 내부에는 Displayer가 없으므로 GUI기반인 Firefox를 바로 실행할 수 없다. 그러므로 xauth를 사용하여 Docker 밖 Host OS로부터 가상모니터를 받아와서 표시한다.

우선 xauth를 설치한다.
apt-get install xauth

Docker 밖의 Host OS에서 xauth list 명령으로 Cookie 값을 알아낸다.



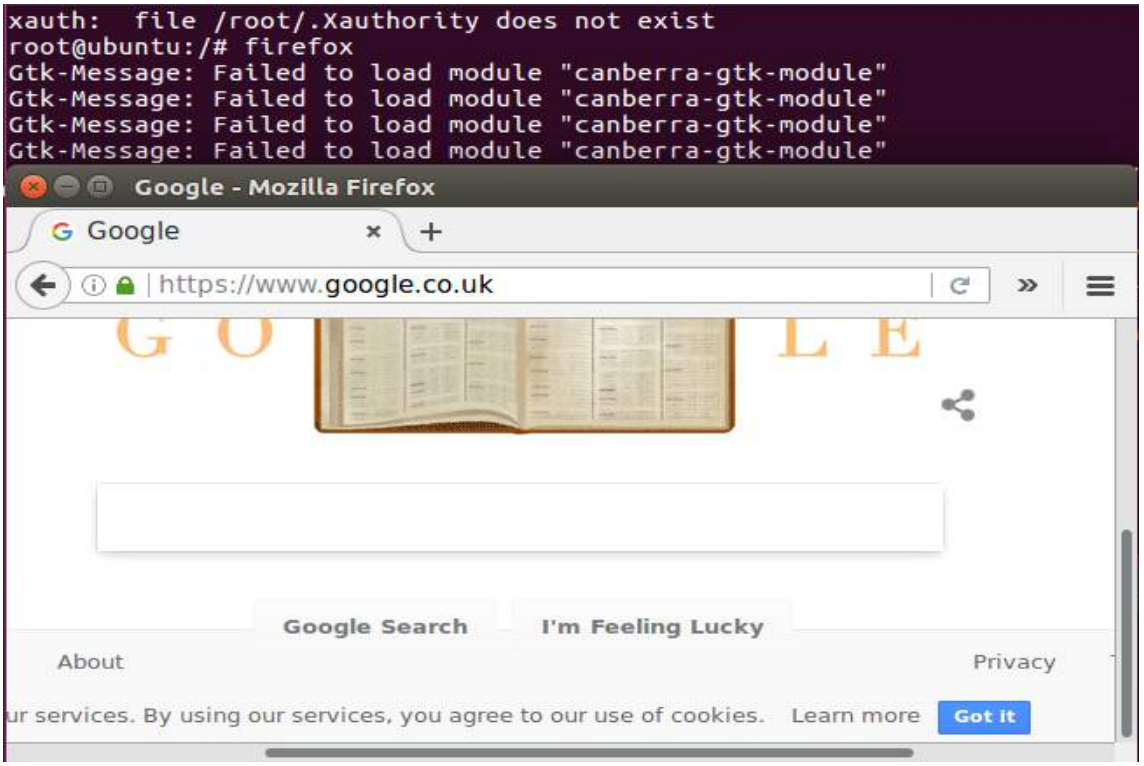
[그림-10] Cookie값

Docker 내부의 Container에 xauth add 명령으로 알아낸 Cookie 값을 추가한다.

```
root@37f8146c3efc:/# xauth add ubuntu/unix:0  MIT-MAGIC-COOKIE-1  3f1fa4ba6514
6cbb5bd1938fcf666ccd
xauth:  file /root/.Xauthority does not exist
```

[그림-11] Cookie 값 추가

이렇게 설정 후 Firefox를 실행하면 Docker 내에서도 정상적으로 GUI의 사용이 가능하다



[그림-12] Docker내에서 firefox가 설치 된 모습

attach 또는 exec 명령을 사용하면 Container에 다시 접속할 수 있다.
sudo docker attach (name or container id)
sudo docker exec (name or container id)

8.3.3 Grafana 및 InfluxDB 설치

o Grafana 설치

1) 도커를 이용하지 않은 설치 방법

```
sudo apt-get update
sudo apt-get install grafana

grafana-server 서비스를 시작한다.
sudo service grafana-server start
```

2) 도커를 활용한 설치 방법
 다음 명령을 이용하여 -d옵션으로 Grafana를 백그라운드 모드로 설치후 실행한다.
        docker run -d grafana/grafana

-이미지파일 뒤에 특정 version을 입력하여 설치 할 수 있다.
        grafana/grafana:version

-p옵션을 이용하여 Grafana기본포트(3000)를 열수 있다.
        -p 3000:3000

-v옵션을 이용하여 볼륨 설정을 한다.
        -v /var/lib/grafana:/var/lib/grafana

-e옵션을 이용하여 Grafana내부의 옵션 및 패널을 설정후 설치할 수 있다.
        -e “GF_SECURITY_ADMIN_PASSWORD=secret“
        -e “GF_INSTALL_PLUGINS=grafana-clock-panel“

--name 옵션을 이용하여 도커컨테이너의 이름을 설정후 설치 할 수 있다.
        --name = granfa

o InfluxDB 설치
1) 도커를 활용하지 않은 설치 방법
 다음 명령을 사용하여 InfluxData 저장소를 추가한다.
        curl -sL https://repos.influxdata.com/InfluxDB.key ¦ sudo apt-key add -
source /etc/lsb-release
        echo“debhttps://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME}
stable“ ¦ sudo tee /etc/apt/sources.list.d/InfluxDB.list

        InfluxDB를 설치한다.
        sudo apt-get update
        sudo apt-get install InfluxDB

        InfluxDB 서비스를 시작한다.
        sudo service InfluxDB start

2) 도커를 활용한 설치 방법

다음 명령을 이용하여 InfluxDB 포트인 8086을 열고 -v옵션으로 볼륨설정을 하고 설치 후 실행한다.

- docker run InfluxDB

-v옵션을 이용하여 볼륨 설정을 한다.

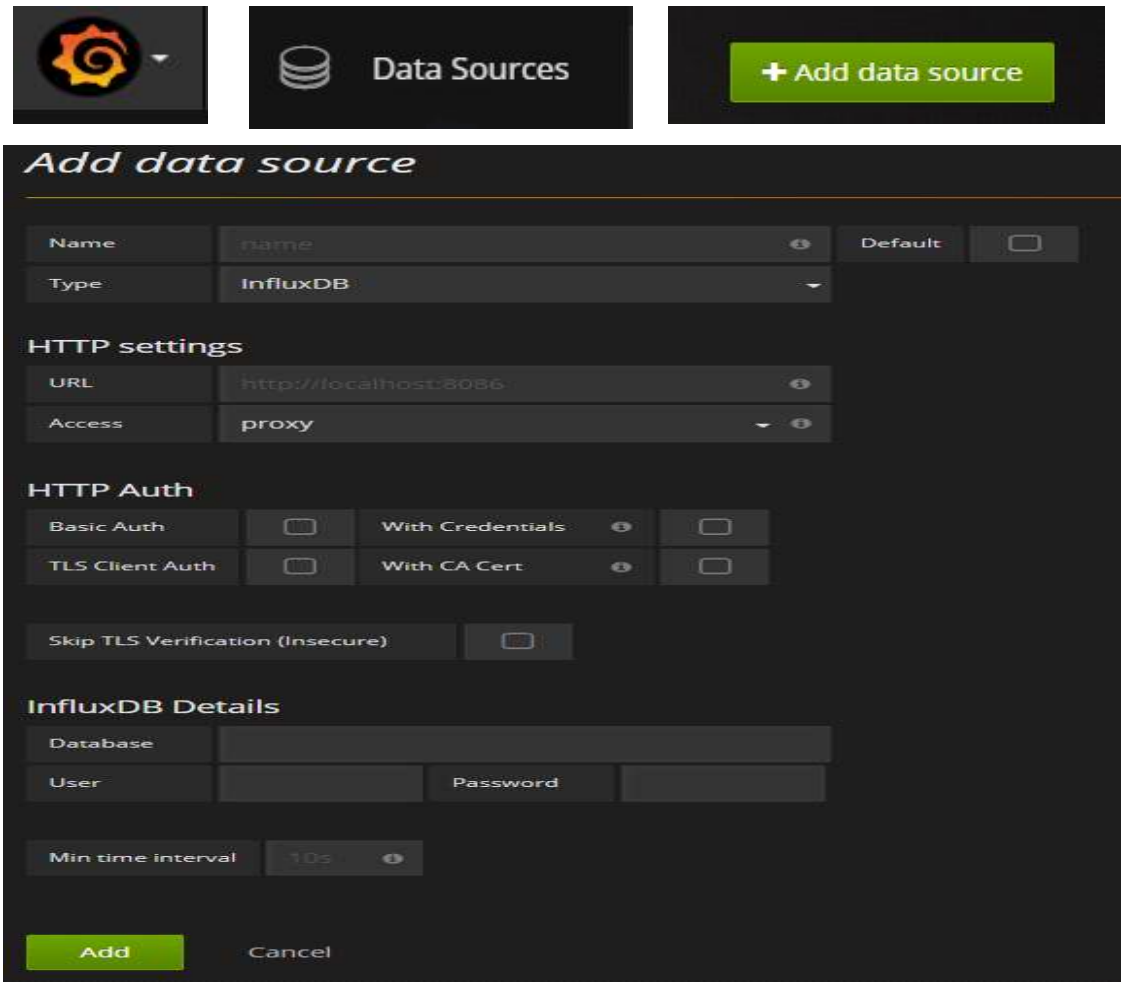
-v $PWD:/var/lib/InfluxDB

-p옵션을 이용하여 InfluxDB기본포트(8086)를 열수 있다.

-p 8086:8086

8.3.4 Grafana 설정 방법

o InfluxDB 데이터 가져오는 방법



[그림-13] 데이터베이스 설정 값 입력

상단에 그림 순서대로 클릭을 한 후 Add data source 항목에서 InfluxDB에 설정 정보를 토대로 작성한다.

[그림-14] 데이터베이스 추가 성공

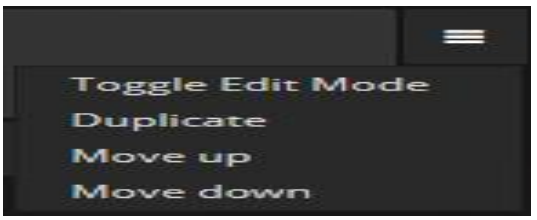
작성이 끝난후 Add버튼을 클릭한다. 문제가 없다면 Data source is working이라는 메시지가 출력된다.

o 패널에 데이터를 추가하는 방법

[그림-15] 패널에 Metric을 추가하는 방법

패널 상단을 클릭한 후 Edit버튼을 클릭한다. 그러면 패널 하단에 탭이 생성된다. 각각의 탭은 패널별로 상이하다. Metric탭을 클릭한 후 Data Source의 default항목을 InfluxDB정보를 추가 한 데이터베이스 이름으로 변경한다.

[그림-16] Query문 작성 방법 (1)



[그림-17] Query문 작성 방법 변경

[그림-18] Query문 작성 방법 (2)

[그림 19]를 보면 해당 Metric에 맞는 Query문을 선택하는 방법과 그림 [그림 20]에서 Toggle Edit Mode를 클릭한 후 그림 [그림 21]과 같이 Query문을 직접 작성하는 방법이 있다. 이 방법들을 이용하여 필요한 데이터들을 패널에 표시 할 수 있다.

o 패널 Plugin 설치방법

1) 도커를 활용하지 않은 설치 방법
https://grafana.com/plugins/ 그라파나 plugin페이지에 접속하여 원하는 패널을 선택한다. 선택 후 리눅스 터미널에 해당 패널에 맞는 설치 명령어를 입력한다.

```
grafana-cli plugins install grafana-piechart-panel
```

[그림-19] 예)Pie Chart 설치 명령어

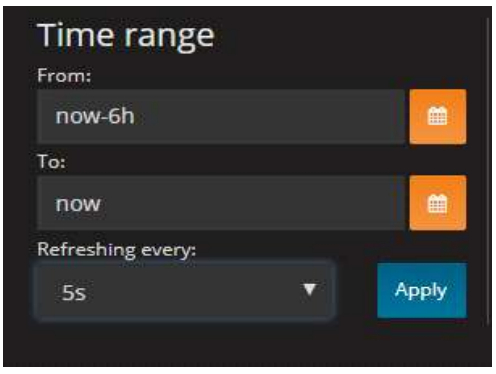
2) 도커를 활용한 설치 방법
도커를 활용한 Plugin설치 방법은 도커로 Grafana를 run하기 전 plugin을 추가하는 옵션을 추가하여 해당 plugin을 설치한다.

```
-e "GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-simple-json-datasource"
```

[그림-20] 예)도커를 이용한 Clock 패널 설치 옵션

o 패널 시간 범위 및 갱신 시간 설정
From과 To를 수정하여 시간을 설정 할 수 있다.
Refreshing every항목을 설정하여 설정한 시간 마다 그래프가 갱신되게 할 수 있다.

o Time Shift 설정 방법
From과 To에 now(현재시간)을 기준으로 d(일), h(시간), m(분), s(초)를 설정하여 그래프가 시간에 흐름에 따라 변화하는 것을 확인 할 수 있다.



[그림-21] Time range 설정 탭

[그림-22] Time range를 설정한 패널

o 패널 iframe Url 확인 방법

그라파나 패널의 iframe을 사용하기 위한 방법이다. 먼저 패널 상단부를 클릭한 후 Share를 클릭한다. 하단 오른쪽 그림에서 Embed항목을 클릭하면 해당 패널의 iframe을 확인 할 수 있다.

이때 주의점은 패널에서 Time Shift를 적용시켜도 iframe에서는 적용 되지 않는다는 것이다. 이유는 iframe을 가져오는 시점의 시간이 저장되기 때문이다. 이러한 문제점을 해결하기 위하여 iframe 속성 안에 있는 to 와 from을 Date().getTime으로 설정하여 현재시간을 동적으로 받아오는 코드를 작성한다.

[그림-23] 패널 iframe 확인 방법

8.3.5 예제 Data pump

python에서 InfluxDB를 import 함으로써 InfluxDB의 Query문을 제어 할 수 있다.
먼저 pip install InfluxDB 명령어를 이용하여 InfluxDB 모듈을 설치한다. 다음은 InfluxDB 모듈을 이용한 Data pump 코드의 예시다.

JNU_pump.py

```
#!/usr/bin/env python
#coding=utf-8
from datetime import datetime, timedelta
import pprint
from InfluxDB import InfluxDBClient
from copy import deepcopy
import pytz
import random
import time
import sys
from JNU_data import c_messageJSON

local_tz = pytz.timezone('Asia/Seoul') # use your local timezone name here

def utc_to_local(utc_dt):
        local_dt = utc_dt.replace(tzinfo=pytz.utc).astimezone(local_tz)
        return local_tz.normalize(local_dt) # .normalize might be unnecessary

def get_ifdb(db, host='localhost', port=8086, user='root', passwd='root'):
        client = InfluxDBClient(host, port, user, passwd, db)
        try:
                client.create_database(db)
        except:
                pass
        return client

def my_test(ifdb):
        tablename = "JNU_2_0"
        deviceid="JNU_Device"
        messageJSON = c_messageJSON(deviceid)

        json_body = [
        ]
        status= 'status'
        lat='lat'
        lon='lon'
        ratio = 'ratio'
        concentration = 'concentration'
        low_pulse_occupancy = 'low_pulse_occupancy'
        alcohol_gas ='alcohol_gas'
        co_gas ='co_gas'
        temp='temp'
        light='light'
```

```
        point = {
                "measurement": tablename,
                "tags": {
                        "host":messageJSON.hostname,
                        "device_ID":messageJSON.deviceid,
                },
                "fields": {
                        status :messageJSON.status,
                        lat :messageJSON.lat,
                        lon:messageJSON.lon,
                        ratio:messageJSON.ratio,
                        concentration:messageJSON.concentration,
                        low_pulse_occupancy :
messageJSON.low_pulse_occupancy,
                        alcohol_gas:messageJSON.alcohol_gas,
                        co_gas:messageJSON.co_gas,
                        temp:messageJSON.temp,
                        light:messageJSON.light,
                },
                "time": None,
        }
        while True:
                messageJSON.random_data()
                dt = datetime.now()
                ldt = utc_to_local(dt)
                print "UTC now=<%s> local now=<%s>" % (dt, ldt)
                np = deepcopy(point)
                np['fields'][lat] =messageJSON.lat
                np['fields'][lon] =messageJSON.lon
                np['fields'][status] =messageJSON.status
                np['fields'][ratio]=messageJSON.ratio

np['fields'][concentration]=messageJSON.concentration
np['fields'][low_pulse_occupancy]=messageJSON.low_pulse_occupancy
                np['fields'][alcohol_gas]=messageJSON.alcohol_gas
                np['fields'][co_gas]=messageJSON.co_gas
                np['fields'][temp]=messageJSON.temp
                np['fields'][light]=messageJSON.light
                np['time'] = dt
                json_body.append(np)
                ifdb.write_points(json_body)
                result = ifdb.query('select * from %s' % tablename)
                time.sleep(2)

def do_test():
        ifdb = get_ifdb(db='resource_2_0')
        my_test(ifdb)

if __name__ == '__main__':
        do_test()
```

JNU_data.py

```
import random

class c_messageJSON():
deviceid="JNU_Device";lat=0;lon=0 ;status=0;ratio=0;concentration=
0;low_pulse_occupancy=0;alcohol_gas=0;co_gas=0;temp=0;
light=0;hostname="JNU";

def __init__(self,deviceid):
self.deviceid = deviceid

def
setData(self,lat,lon,temp,ratio,concentration,low_pulse_occupancy,alcohol_g
as,co_gas,light):
self.lat = round(lat,1)
self.lon= round(lon,1)
self.temp=round(temp,1)
        self.ratio = round(ratio,1)
        self.concentration  = round(concentration,1)
        self.low_pulse_occupancy = round(low_pulse_occupancy,0)
        self.alcohol_gas =round(alcohol_gas,0)
        self.co_gas = round(co_gas,0)
        self.light = round(light,0)

def random_data(self):
self.lat = round(random.uniform(-90.0,90.0),1)
self.lon = round(random.uniform(-180.0,190.0),1)
self.temp=round(random.uniform(-40.0,125.0),1)
self.light=round(random.uniform(0,1023),0)
        self.ratio = round(random.uniform(0.0,100.0),1)
        self.concentration = round(random.uniform(0.0,28000.0),1)
        self.low_pulse_occupancy = round(random.uniform(0,30000000),0)
        self.alcohol_gas = round(random.uniform(0,1023),0)
        self.co_gas = round(random.uniform(0,1023),0)
        self.status = round(random.uniform(0,2),0)
```

8.4 결론

다양한 사물들이 협업하는 사물인터넷에서 관리 시스템의 구축에 가장 큰 걸림돌은 개별 사물들이 너무 많고, 제공하는 정보가 너무 다양하다는 것이었다. 본 기술에서는 이러한 문제점을 효과적으로 해결하기 위하여 Grafana에서 개별 센서에 대응되는 위젯을 설계하고, 해당 위젯을 하나의 대시보드에서 조립하여 사물인터넷용 대시보드를 개발하였다. 본 기술은 드론 센서와 스마트폰 센서를 결합하여 실증하였다.