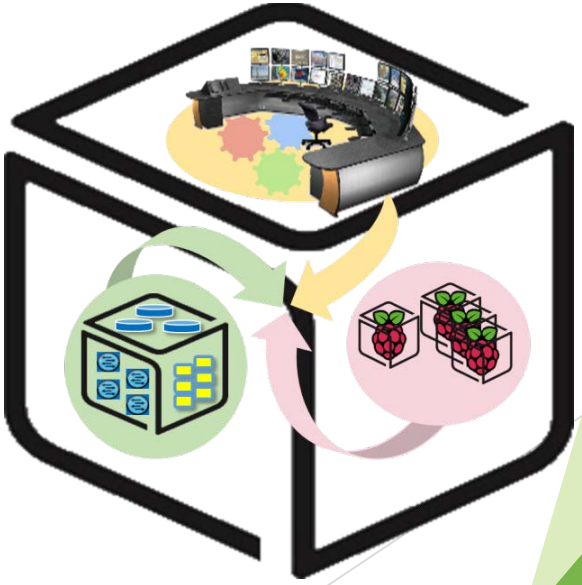
SmartX Labs
for Computer Systems

Tower Lab

(2018, Spring)



NetCS Lab

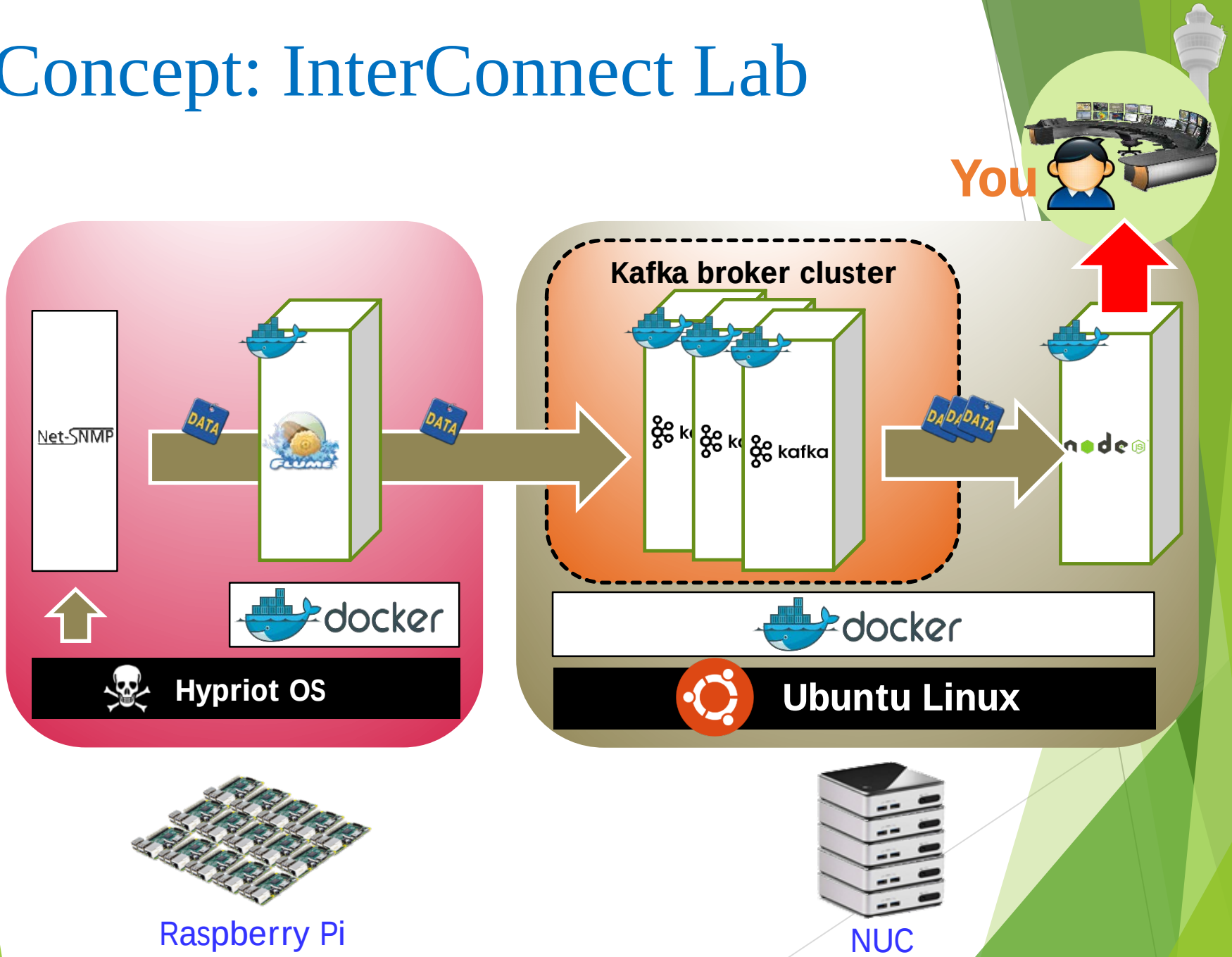
History and Contributor of Tower Lab (2018 Spring)

Version	Updated Date	Updated Contents	Contributor
-	2016/04	(구) Playground Lab 최종본 작성	김 병 돈
v1.0	2016/05	Build Lab 초안 작성 (Outline 및 Control Tower 추가)	김 승 룡
v1.1	2016/05	검수 및 제안 사항 반영 (apt-get source 변경 내용 외)	김 승 룡 김 진 우
v1.2	2016/05	NUC에 대한 모니터링 추가 순서 변경에 따른 내용 변경	김 승 룡
v1.3	2016/05	Lab 명칭 변경 (Build -> Tower) Hypervisor 관련 오류 수정	김 승 룡
v2.1	2017/05	Merged with InterConnect & Tower Lab	김 승 룡
v2.2	2018/01	InfluxDB 버전 업데이트 스크립트 변경	이 승 형
V3.0	2018/05	Lab 분할 및 소프트웨어 업데이트에 따른 내용 수정	김 승 룡

Goals

- Understanding Concepts
  - InterConnect, Tower
  - MSA(Micro Service Architecture)
  - TSDB(Time Series Database)
  - Visibility & Visualization
- Visualization of Resource Visibility
- Monitor and Control

Concept: InterConnect Lab



Concept: Tower Lab

Control Tower

Visibility Center (**Container**)

Orchestration Center (Container)

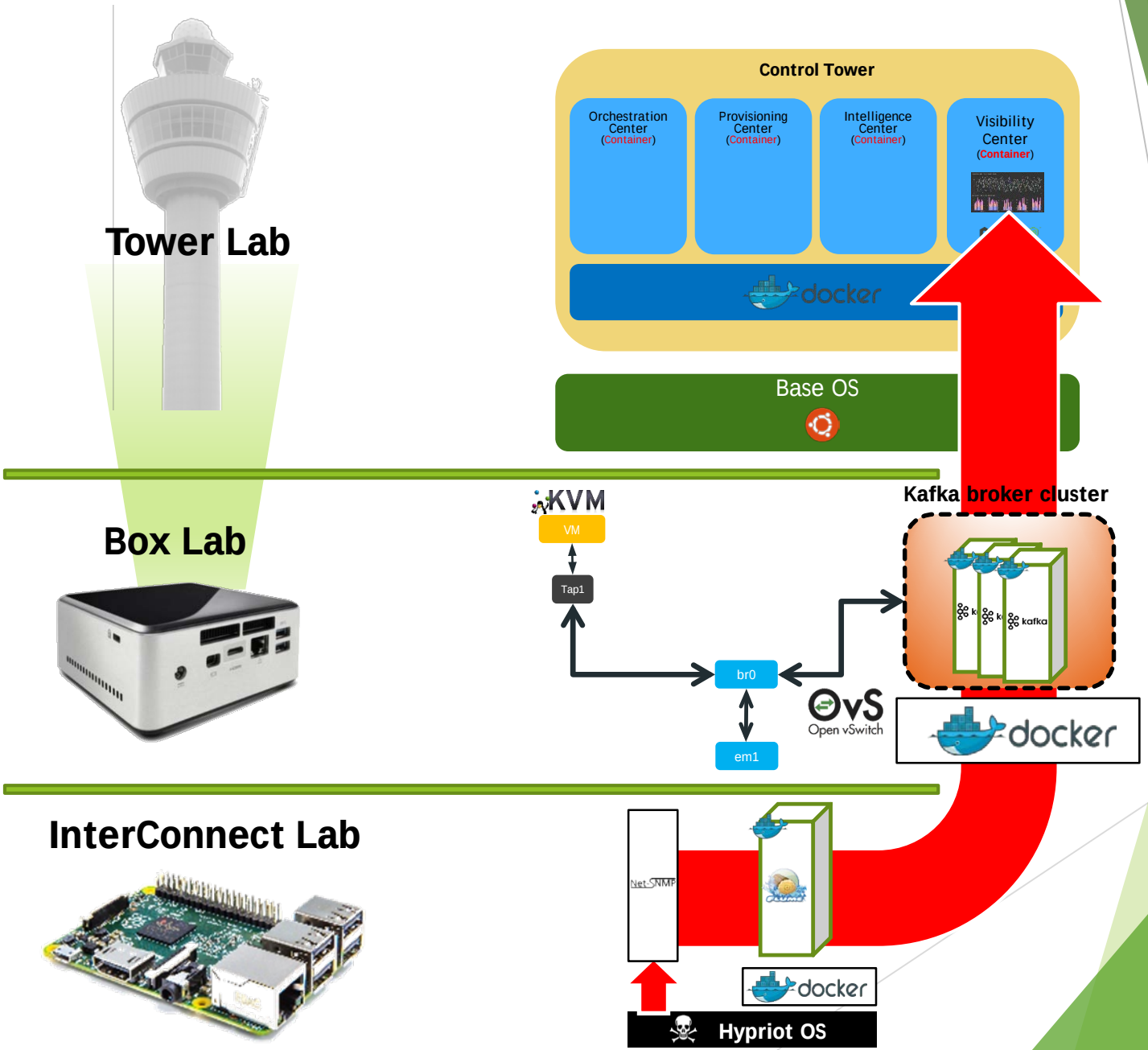
Provisioning Center (Container)

Intelligence Center (Container)

docker

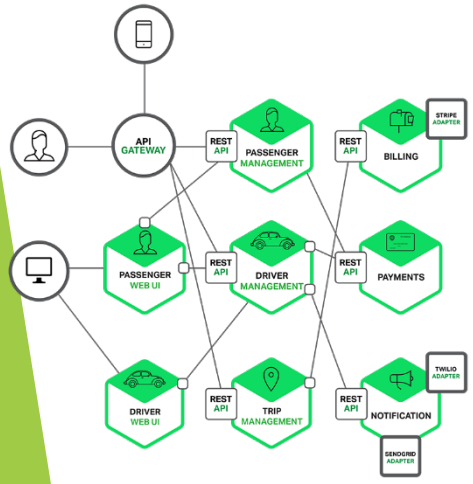
Base OS

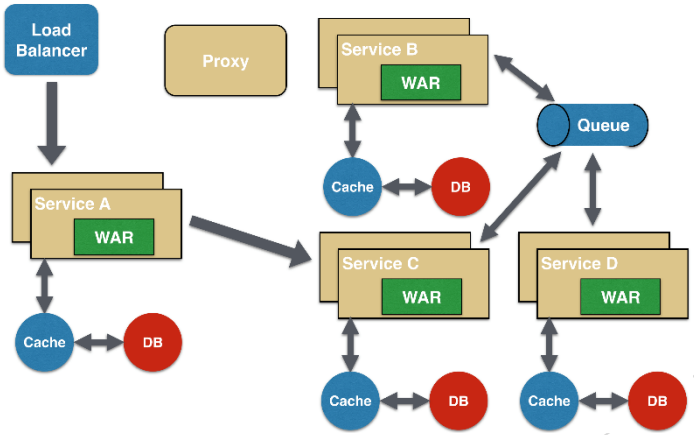
Relation of SmartX Lab

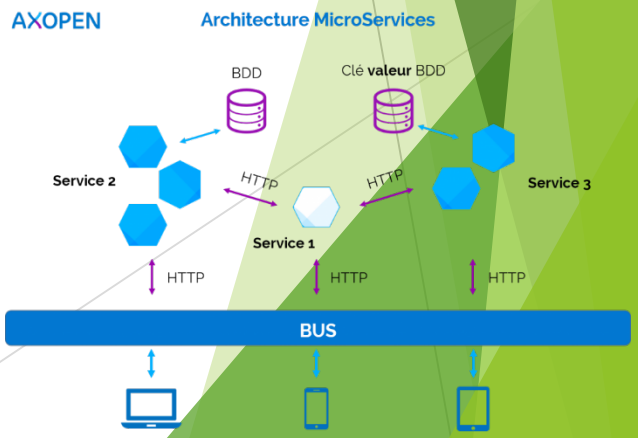


MSA(Micro Service Architecture)

- Software development technique
- Collection of loosely coupled services.
- fine-grained services and lightweight protocols
- Improves modularity
- Makes the application easier
- More resilient to architecture erosion







TSDB(Time Series Database)

- Software system that is optimized for handling time series data, arrays of numbers indexed by time.
- In some fields these time series are called profiles, curves, or traces.
- A time series of stock prices might be called a price curve.
- A time series of energy consumption might be called a load profile. A log of temperature values over time might be called a temperature trace.

Visualization of Resource Visibility

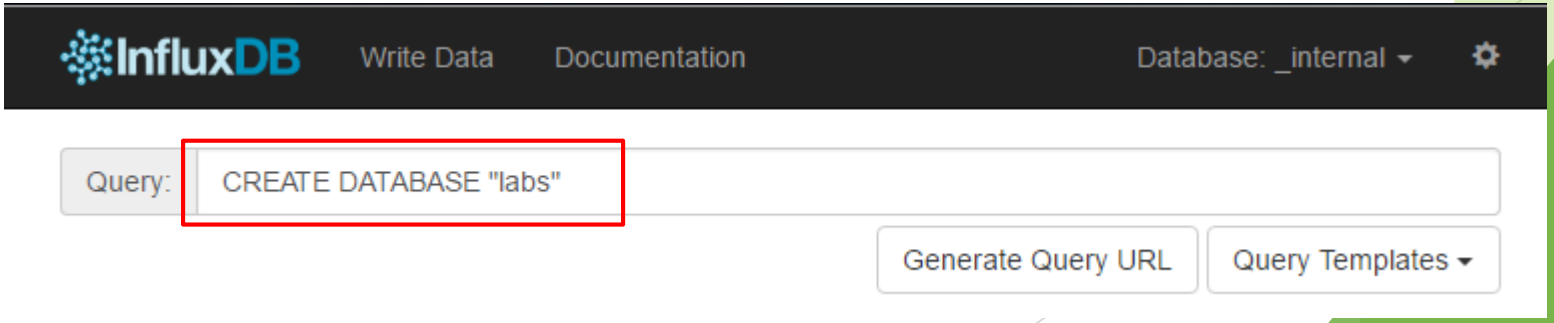
0. Finish InterConnect Lab

- Check List
  1. NUC
     - Zookeeper Container
     - Kafka Containers (3 Containers)
     - Consumer Container
  2. Raspberry Pi
     - Flume Container

- Are they working? If you can see logs of resource status on console consumer, go ahead!

1. Run InfluxDB Container on NUC

- Run InfluxDB Container
  - $ cd SmartX-mini/ubuntu-influx
  - $ docker run -p 8083:8083 -p 8086:8086 -v $PWD:/var/lib/influxdb -e INFLUXDB_ADMIN_ENABLED=true --name influx influxdb:1.0
- Connect Web UI
  - http://localhost:8083/
- Create Database(Name: labs)
  - CREATE DATABSE “labs”



2. Insert Data from Kafka to InfluxDB

- Modify hostname in source code ‘kafka_to_db.js’
  - $ cd SmartX-mini/ubuntu-kafkatodb
  - $ vi kafka_to_db.js

```
// InfluxDB
var DB = new influx.InfluxDB({
    // single-host configuration
    host: 'nuc',
    port: 8086, // optional, default 8086
    protocol: 'http', // optional, default 'http'
    username: 'admin',
    password: 'admin',
    database: 'labs'
});

var resourceKafka = new kafka.Client('nuc:2181');
var resourceOffset = new kafka.Offset(resourceKafka);
```

- Build and run container
  - $ sudo docker build --tag kafkatodb .
  - $ sudo docker run -d --net=host --name kafkatodb kafkatodb

3. Check Data in InfluxDB

- Then, we can check the data in DB
  - SELECT * FROM resource

Query: select * from resource

Generate Query URL | Query Templates

resource

time	cp	cpu	deviceId	disk	ip	memory	rx	rxDropped	rxError	timestamp	tx	txDropped	txError
2017-05-01T13:26:59.922240539Z	"iot"	0.09	"rpi82"	7	"203.237.53.82"	79740	386319858	0	0	"1493645219899"	21085854	0	0
2017-05-01T13:27:00.168460055Z	"iot"	0.43	"rpi88"	1	"203.237.53.88"	9867732	1940079932	0	0	"1493645220163"	50921540	0	0

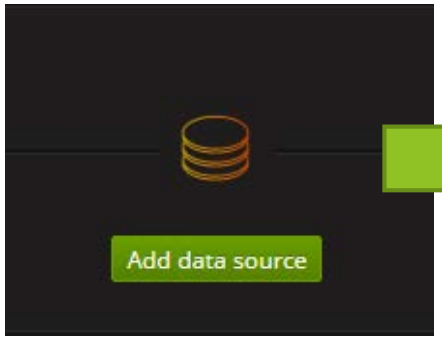
4. Run Grafana Container

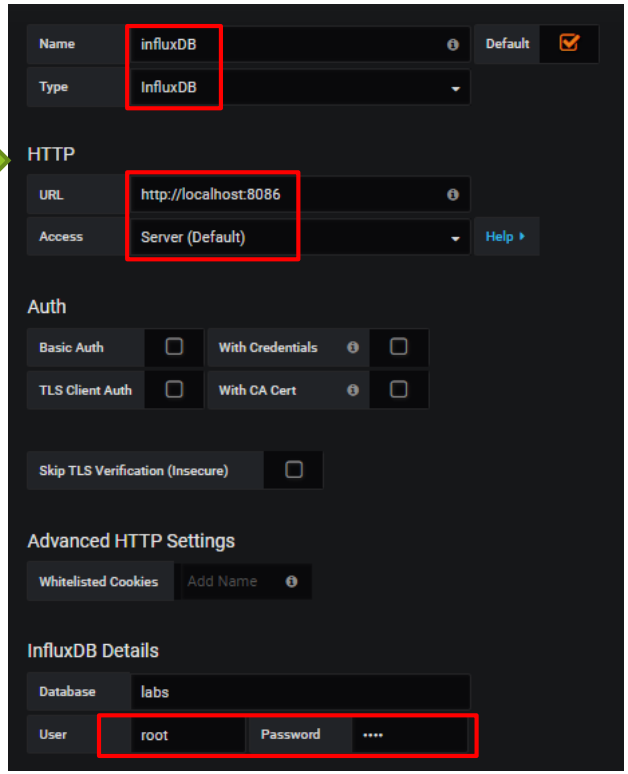
- Run InfluxDB Container
  - $ docker run -d --net=host --name grafana grafana/grafana

- Connect Web UI
  - http://localhost:3000/
  - ID/PW = admin/admin

5. Configure Grafana Dashboard

- Follow below sequences with written option values

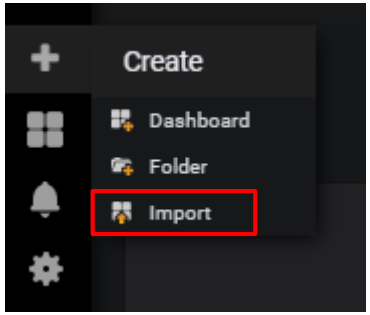


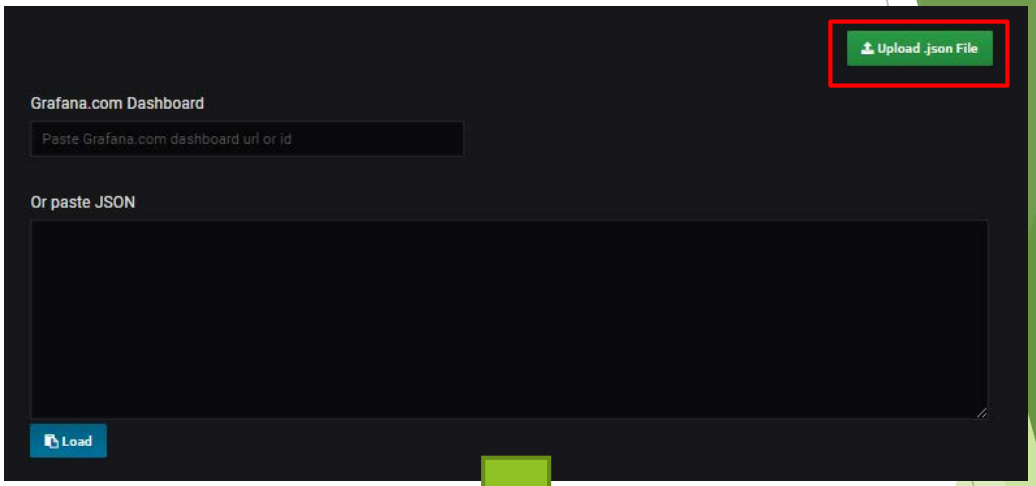


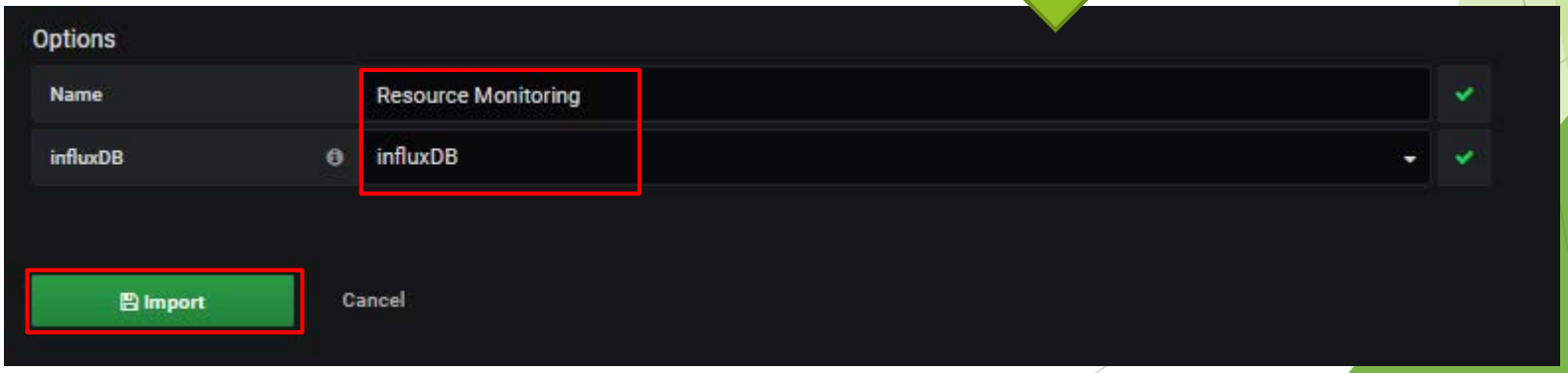
root / root

5. Configure Grafana Dashboard

- Follow below sequences with written option values

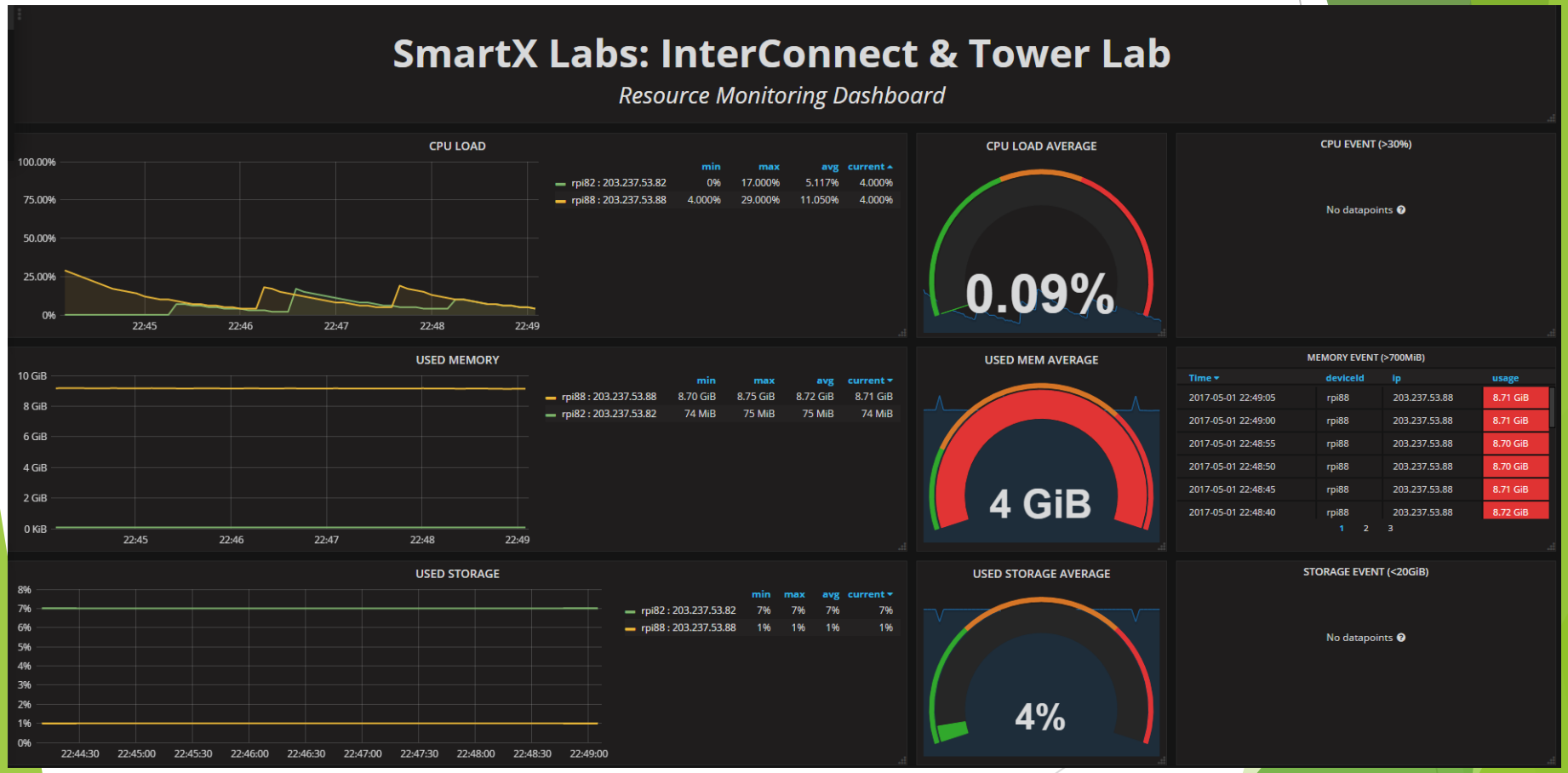






6. Check Dashboard

- We can see the changes of values from database



7. Stress Test

- Install and start a stress test tool
  - $ sudo apt-get install stress
  - $ stress -c 4 -t 60s
- What happens on the dashboard?
- Why?
- How can we monitor metrics faster and more reliable?

Thank You for Your Attention Any Questions?

(참고)
Container 변경사항 저장 및 재시작

- Commit Container
  - 컨테이너 내의 변경사항을 반영하여 새로운 컨테이너 이미지 작성
  - Ctrl+P+Q
  - docker commit -a “[username]” -m "add visualization server based node.js" visualization visualization:0.1

```
srkim@ubuntu:~$ docker commit -a "srkim" -m "add visualization server based node.js" visualization visualization:0.1
sha256:b5ca7015908b7438e1d47f372ab0b03627baed08fa1f8e11c88366f0c1c3dfda
srkim@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
visualization       0.1                 b5ca7015908b        4 seconds ago       325 MB
<none>              <none>              867c578dd875        58 seconds ago      325 MB
ubuntu              14.04               8fa7f61732d6        5 days ago          188 MB
```

- Restart Container
  - Stop했던 컨테이너를 Restart하면 이전 작업 내용을 유지한 채로 다시 컨테이너를 시작할 수 있다.
  - docker stop visualization
  - docker restart visualization