+

Docker Microservices with Informix

Docker & Informix

Hands-on Lab
Version 1.0
Updated: 2019-09-03

DARIN TRACY

HCL TECHNOLOGIES

TABLE OF CONTENTS

1 LAB 1 INSTALL DOCKER & DOCKER-COMPOSE IN VIRTUAL BOX VM.

This lab contains steps in setting up a VirtualBox VM and installing docker and docker-compose. These will be used for the remainling of the labs.

Virtual box is an easy way to run a linux environment on most laptops and perform the functions that you would with a typical linux installation.

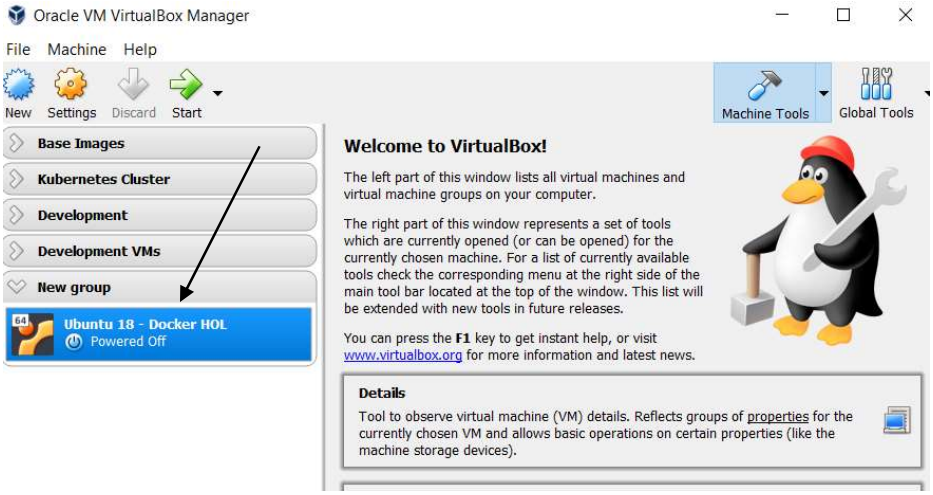
You will install docker and docker-compose into your virtualbox VM environment. You will download docker images that will be used throughout the remaining labs.

The Informix Docker image will use the Informix Developer Edition and in doing this lab you agree to all licensing requirements here. License

This lab uses a Virtualbox VM to run docker. Docker images are used inside your Virtualbox VM. The image that will be used will need access to the internet to download and update accordingly.

1.1 START VIRTUAL BOX VM – (UBUNTU 18 – DOCKER HOL)

__1. From your laptop start Oracle VM VirtualBox. After you've run Oracle virtualbox you will see a screen like the one below. Find the Ubuntu 18 – Docker HOL image and double click on the image to power it on.



__2. As the Ubuntu Image starts up. You should see something like the screen below. Maximize your window and you should see the Informix login option. Login with the following credentials:

User: **informix**
Password: **informix**

1.2 What is Docker & Docker Compose

A container is a standard unit of software that packages up code and all its dependencies so the application is easy to run and deploy from one computing environment to another.

Docker is the tool that allows you to build a container image. It also allows you to manage containers and images. Storage of these docker images is typically in a docker registry. Dockerhub is a registry (repository) of docker images available for public use, similar to github.

A docker container is typically designed to do one thing well as opposed to a single monolithic application. So for a Solution that has many moving parts you may have multiple containers that to work together.

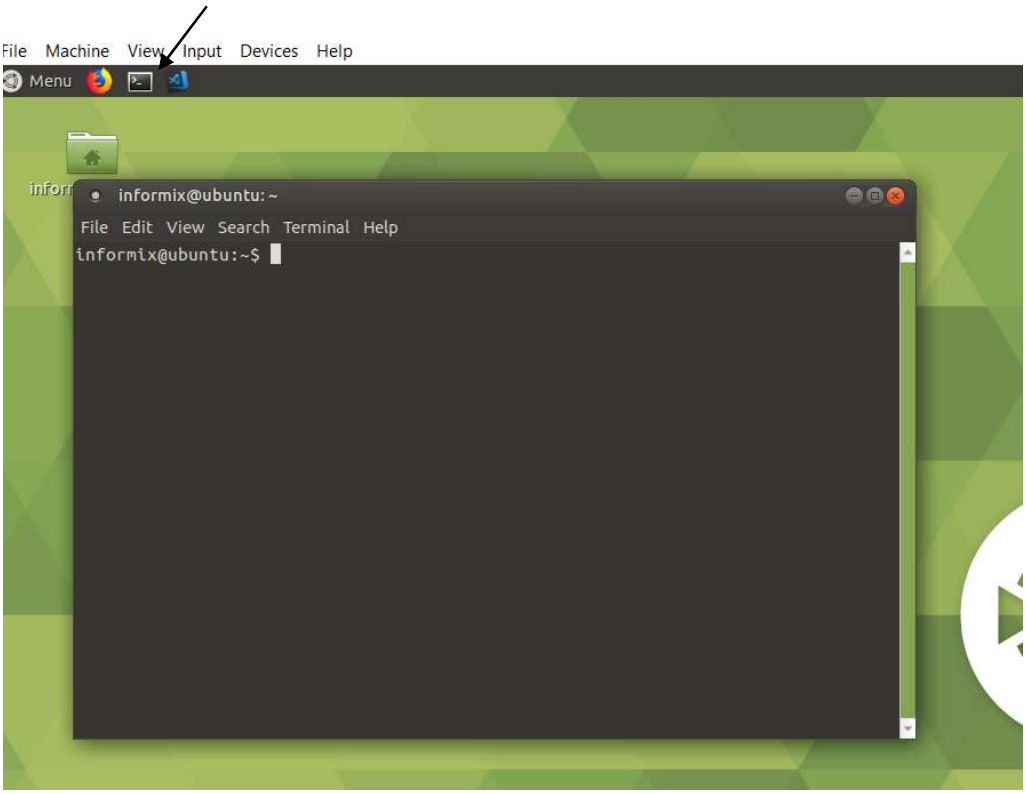
This is where microservices come in. The larger application is broken down into smaller components called microservices. Each single container can be designed, built and mainted to handle one of these smaller components of the larger application.

You can run docker containers manually, create your own network, etc. to allow your group of containers to "work together". But there is an easier method.

Docker-compose allows you to run a group of containers (1 or more) as an entire solution. It will create the networking needed for the containers to communicate with each other. We will be using docker-compose as the vehicle to run our microservices application for this Hands on Lab.

1.3 INSTALL DOCKER & DOCKER COMPOSE

__3. Start a terminal in the Virtualbox VM. Click on the terminal icon in the top panel. This will start a terminal window and log you in as user Informix. We are going to run some commands inside the VM to update it and get it ready for docker.



__4. Switch to user root with the sudo command. You should be able to sudo to root with no password needed in this VM.

```
> sudo bash
```

__5. For future reference the install doc is here. Install Doc has documentation on installing **docker** on Ubuntu 18.

__6. Create a working directory for all work that you do and save.

```
> mkdir /home/informix/lab-<userid>
> cd /home/informix/lab-<userid>
```

__7. Update the VM. In your terminal window run the following.

```
> apt-get update
```

__8. Run the following to install various packages and dependencies:

```
> apt-get install apt-transport-https ca-certificates curl \
  software-properties-common
```

__9. Add Docker's GPG key

```
> curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo apt-key add -
```

__10. Add the Ubuntu repository

```
> add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable"
```

__11. Update and Install docker

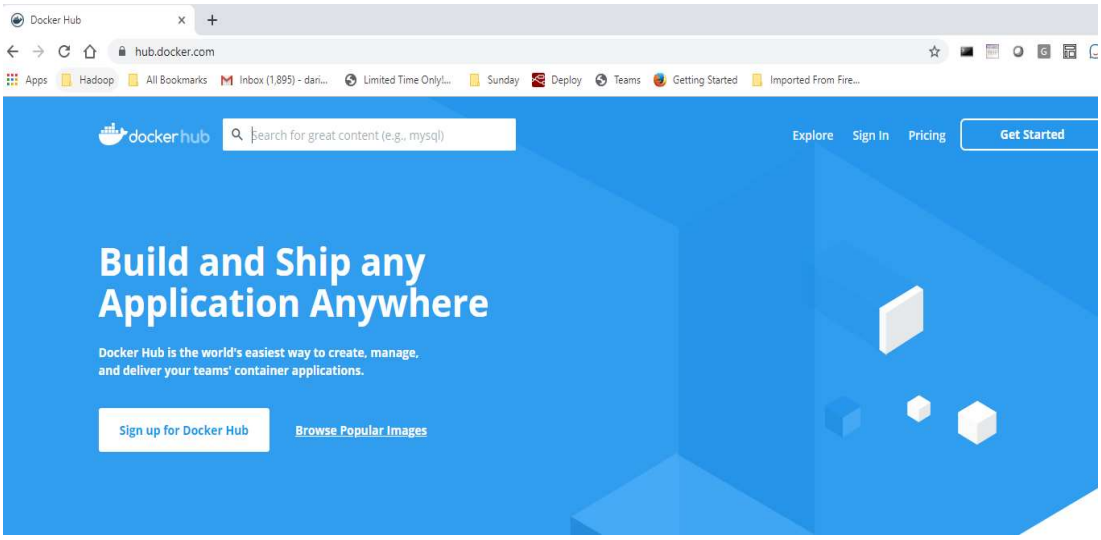
```
> sudo apt-get update
> sudo apt-get install docker-ce
```

__12. Install docker-compose

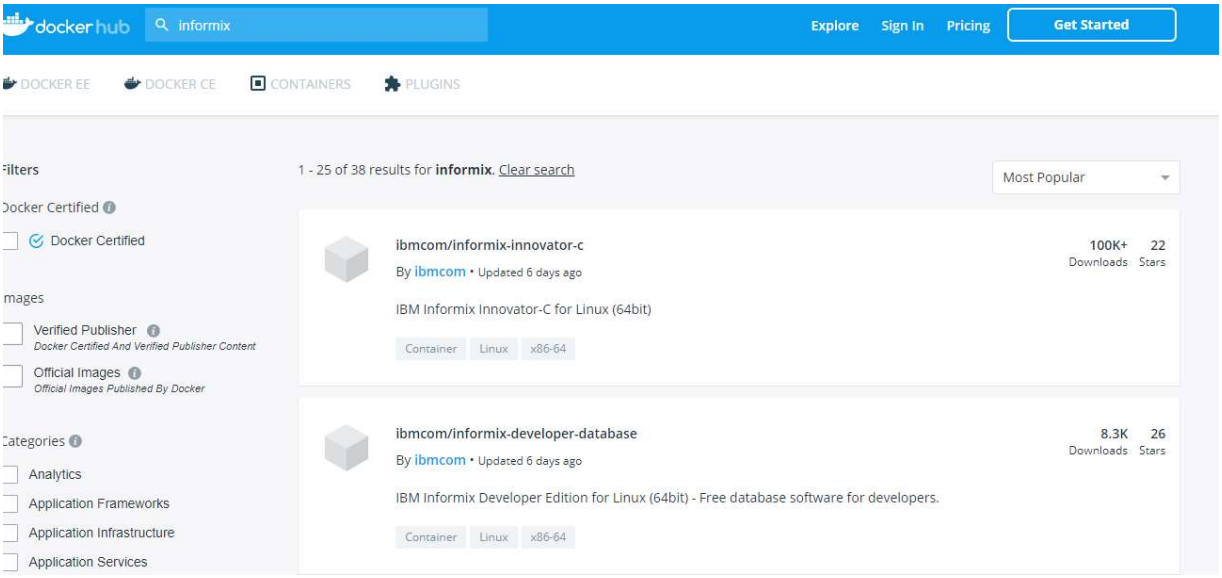
```
> sudo apt-get update
> sudo apt-get install docker-compose
```

1.4 Download your first Docker image(s)

__13. Open a Web browser, on the Windows host or inside the VirtualBox VM. Navigate to http://dockerhub.com



__14. In the search bar Type: informix and hit enter. Click on the ibmcom/informix-developer-database



__15. To pull the docker image run the following command:

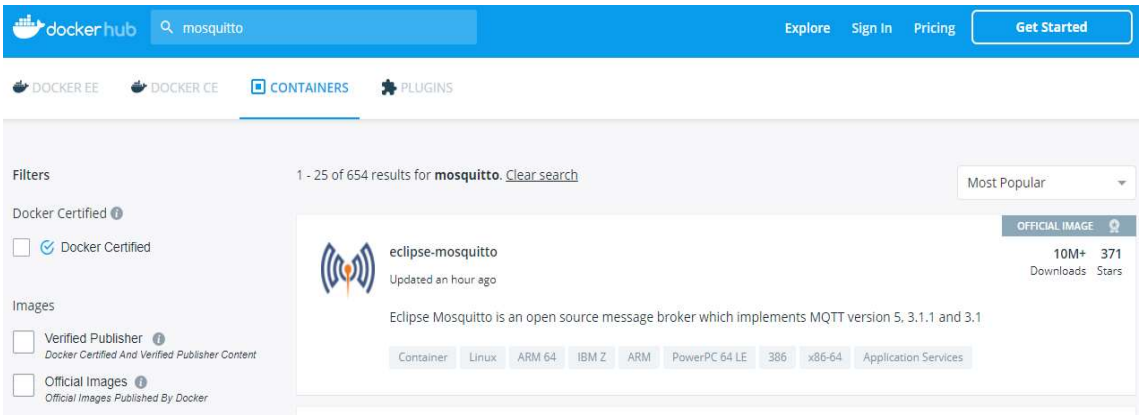
```
> docker pull ibmcom/informix-developer-database
```

__16. We will use two other docker images for this Lab. Download both of these.

__a. Download the python image


```
> docker pull python
```

__b. Download the eclipse/mosquito image




```
> docker pull eclipse-mosquitto
```

__17. Now verify the image were download with the **docker images** command. You should see output like below:

```
root@informix-VirtualBox:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
eclipse-mosquitto                   latest              c0df24a48364        4 days ago          5.75MB
ibmcom/informix-developer-database  latest              3279ba63b237        5 days ago          893MB
python                              3.6-alpine          95a9e476c634        3 weeks ago         95MB
root@informix-VirtualBox:~#
```

2 LAB 2 RUN DOCKER IMAGE

This lab contains the first step in running the docker images you downloaded in the firs lab. We will run our images using docker-run and using docker-compose.

We will download a docker-compose project from github and learn how to run docker containers with docker-compose.

This are different methods of running docker containers as well as running containers that work together. Our preferred method will be docker-compose for this. Breaking work into smaller components and running gultiple containers to do the work for a larger solution is the idea of microservices.

2.1 RUN IBMCOM/INFORMIX-DEVELOPER-DATABASE IMAGE

__18. cd to your working directory.

```
> cd /home/informix/lab-<userid>
```

__19. Create a run.de file using **vi** with the following contents in the file.

```
IMG=ibmcom/informix-devloper-database
docker run -it --name server -h server -e LICENSE=accept \
   -e SIZE=small $IMG
```

__20. In another terminal window (sudo to root) run the docker image.

```
> cd /home/informix/lab-<userid>
> chmod 777 run.de
> ./run.de
```

```
>>>>    ONCONFIG UPDATE: PHYSBUFF - 128
>>>>    ONCONFIG UPDATE: ROOTSIZE - 350000
>>>>    ONCONFIG UPDATE: SINGLE_CPU_VP - 1
        [COMPLETED]
>>>     DISK INITIALIZING (Tue Sep 17 15:06:08 UTC 2019) ...
>>>     Setting sch_init_informix.sql file ...
>>>         Using Small sch_init_informix.sql
        [COMPLETED]
>>>     Updating HOSTNAME in file(s)...
>>>         [/opt/ibm/informix/etc/sqlhosts]
>>>     Updating HOSTNAME in /opt/ibm/informix/etc/sqlhosts
        [COMPLETED]
>>>     DISK INITIALIZING (Tue Sep 17 15:06:08 UTC 2019) ...
>>>     Create MSGPATH file ...
>>>         [/opt/ibm/data/logs/online.log]
        [COMPLETED]
>>>     DISK INITIALIZING (Tue Sep 17 15:06:08 UTC 2019) ...
>>>     Create rootdbs ...
>>>         [/opt/ibm/data/spaces/rootdbs.000]
        [COMPLETED]
>>>     Running informix_custom_install.sh...
>>>     DISK INITIALIZING (Tue Sep 17 15:06:08 UTC 2019) ...
>>>     Informix DISK Initialization ...
```

__21. In the other window run dbash server. This will connect you to the docker image as if you had ssh'd into it. You can now run onstat commands, dbaccess, etc.

```
> dbash server
```

```
root@informix-VirtualBox:~/lab-darint# dbash server
informix@server:~$ onstat -

IBM Informix Dynamic Server Version 14.10.FC2DE -- On-Line -- Up 00:02:50 -- 172852 Kbytes

informix@server:~$
```

dbash is a small shell script that runs **docker exec -it $* bash**

__22. To stop the docker container run the following:

```
> stop.gen server
```

__23. To allow multiple containers to be able to communicate with each other they have to have a network they can use to do this. Docker allows you to setup a network for just such a thing. See the following docker network command to create a bridged network.

```
> docker network create -d bridge my-network
```

__24. When running a docker image you now specify the network to be a part of. See the docker run command below:

```
docker run -it -name server -h server --network=my-network -e
LICENSE=accept -e SIZE=small $IMG
```

__25. Identify below we started two containers. Using -td (runs as a daemon). Then we logged into server2 and pinged server1. If we hadn't used –network=my-network, this would not be possible.

```
root@informix-VirtualBox:~/lab-darint# docker run -td --name server1 -h server1 -e LICENSE=accept
--network=my-network -e SIZE=small ibmcom/informix-developer-database
7a25f84b59386b1f0dae9f869b5a14efcd9a18b7950ad124734da2ea1aa1bef8
root@informix-VirtualBox:~/lab-darint#
root@informix-VirtualBox:~/lab-darint# docker run -td --name server2 -h server2 -e LICENSE=accept
--network=my-network -e SIZE=small ibmcom/informix-developer-database
5727a1fba85b95cc1b1135403d799fa9c6d3110277945de13cd4b034fbb54dee
root@informix-VirtualBox:~/lab-darint#
root@informix-VirtualBox:~/lab-darint# dbash server2
informix@server2:~$ ping server1
PING server1 (172.18.0.2) 56(84) bytes of data.
64 bytes from server1.my-network (172.18.0.2): icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from server1.my-network (172.18.0.2): icmp_seq=2 ttl=64 time=0.039 ms
```

2.2 DOWNLOAD COMPOSE-MICROSERVICES-LAB GITHUB REPOSITORY

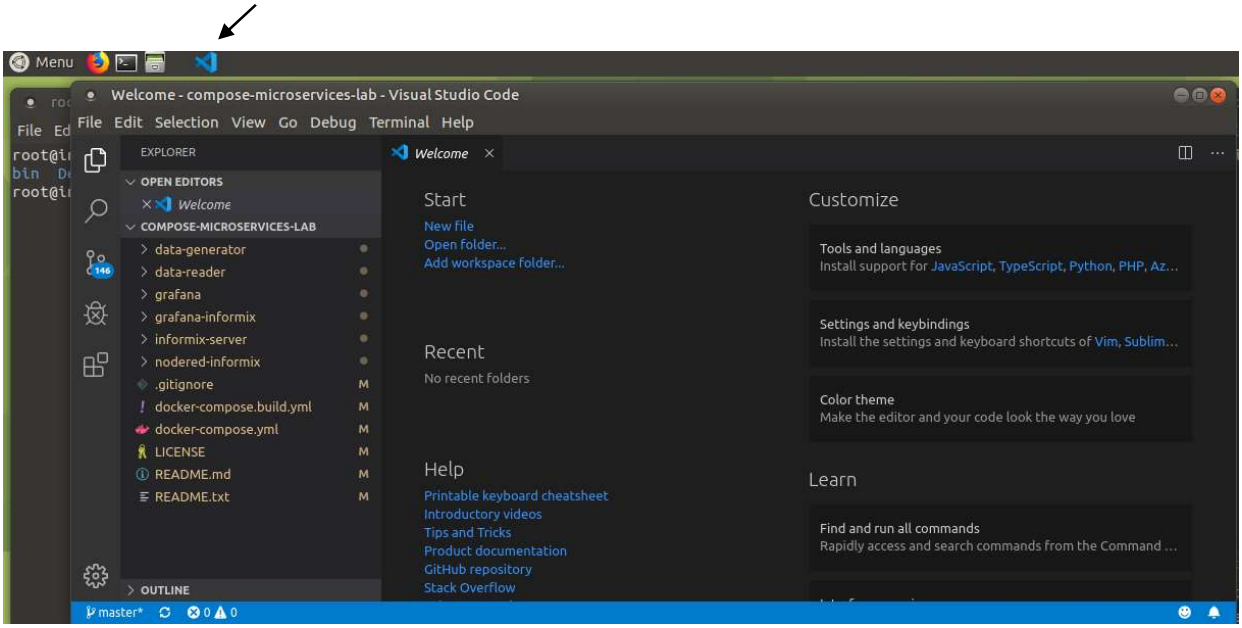
__26. Clone the compose microservices lab from the public github.com website. From your working directory **/home/informix/lab-<userid>** run the following command.

```
> git clone https://github.com/informix/compose-microservices-lab
```

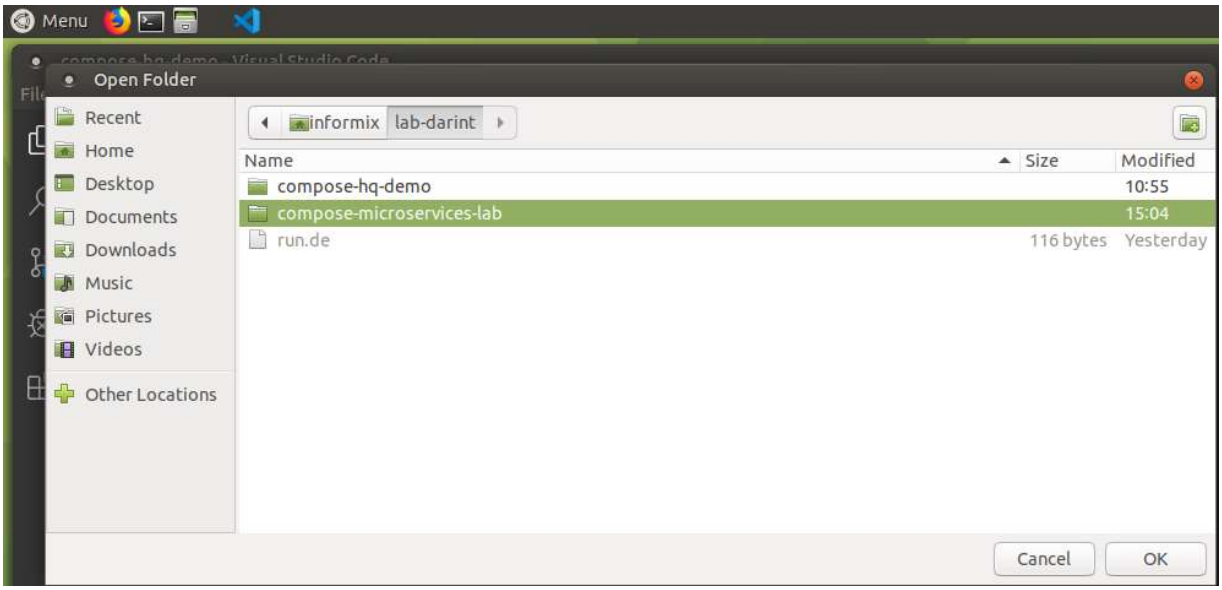
__27. The above command will have downloaded the github project. Run the following command to get ready to use the project.

```
> cd compose-microservices-lab
> chmod -R 777 *
```

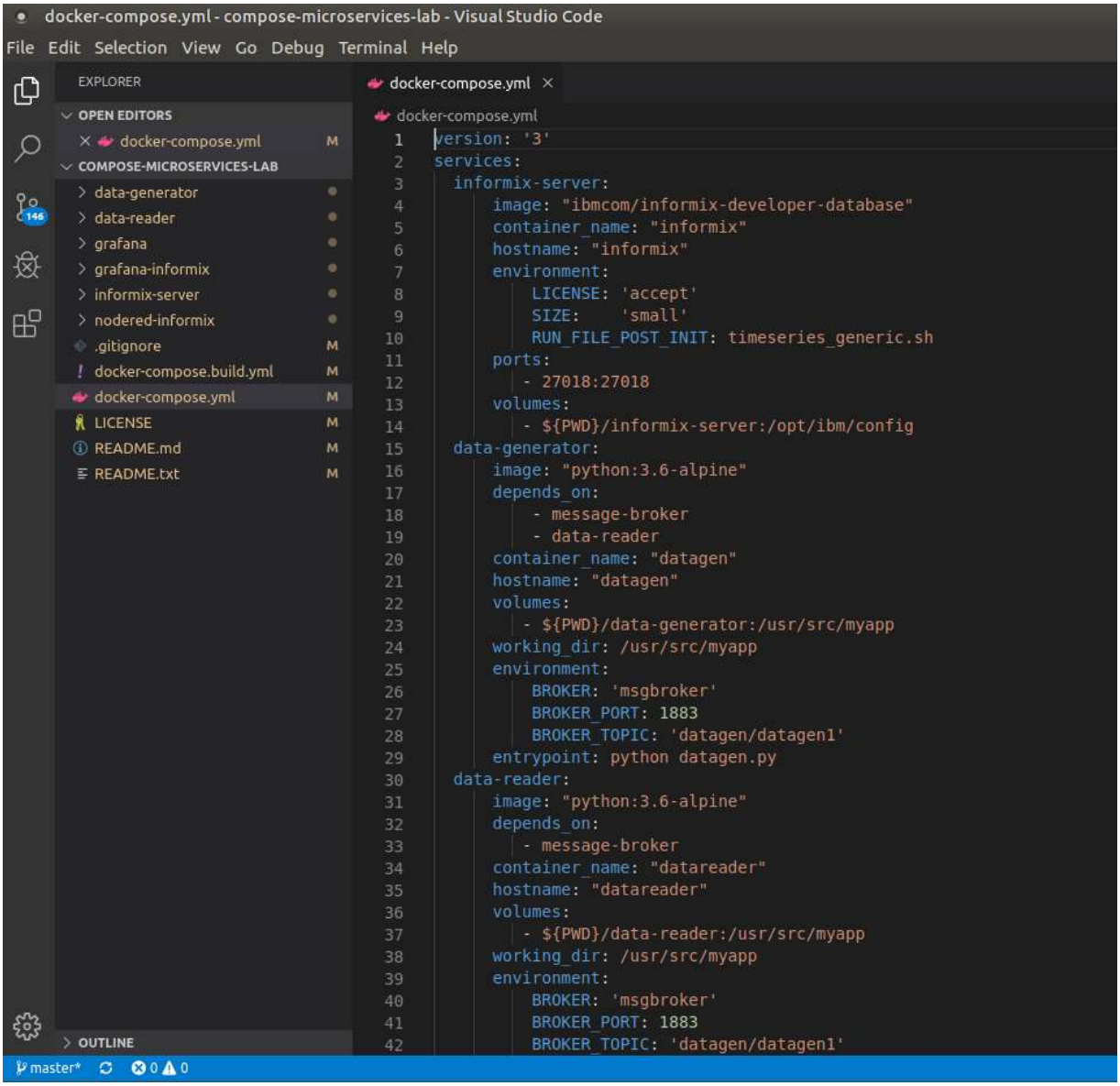
__28. Start VSCode by clicking on the VSCode icon in the top panel.



__29. Choose File -> Open Folder and navigate to your lab-<userid> directory and open the compose-microservices-lab folder.



__30. We will start by opening the docker-compose.yml file. This is the main file that describes what this project will do. What containers run and how they run.

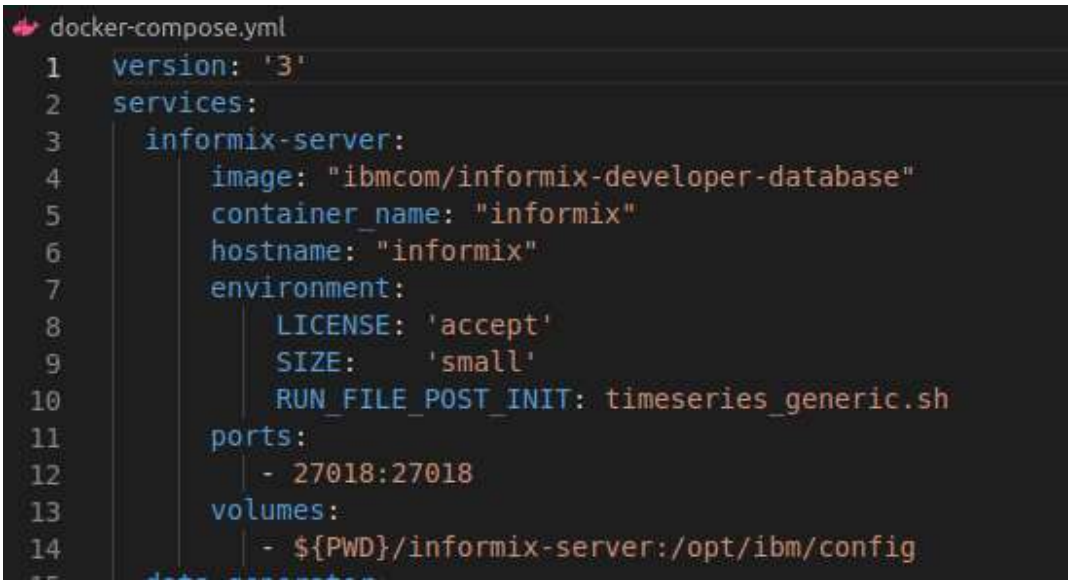


__31. The yaml file list 5 different services. Informix-server, data-generator, data-reader, message-broker and grafana-ui.

__a. Each of these services is a separate container that runs and will be part of the microservices application.

2.3 INFORMIX-SERVER SERVICE

__32. The most important service here is the Informix-server which uses the informix-developer-database docker image on dockerhub.com



__a. The image name of the image is shown here. When we run this for the first time if the docker image is not available on the system it will be downloaded.

__b. We give it a container name and a hostname.

__c. The environment section is used to tell the informix docker image how to run.

__i. LICENSE must be accepted

__ii. We set the SIZE to small

__iii. And we run a shell script to setup a timeseries database/table using the env option RUN_FILE_POST_INIT.

__d. We then expose the port 27018. Which is our default REST port

__e. And we mount a volume with configuration information, which also includes the timeseries_generic.sh/sql files

__i. This gets mounted directly from the project folders into the docker container.

2.4 DATA-GENERATOR SERVICE

__33. The next service here is a basic python docker image. This docker container will run a python program that is supplied externally from a mounted directory. This program will generate random data and publish that data to an **MQTT message broker**.

```
15  data-generator:
16      image: "python:3.6-alpine"
17      depends_on:
18          - message-broker
19          - data-reader
20      container_name: "datagen"
21      hostname: "datagen"
22      volumes:
23          - ${PWD}/data-generator:/usr/src/myapp
24      working_dir: /usr/src/myapp
25      environment:
26          BROKER: 'msgbroker'
27          BROKER_PORT: 1883
28          BROKER_TOPIC: 'datagen/datagen1'
29      entrypoint: python datagen.py
```

__34. This service sets the container name and hostname. Mounts a folder in the project into the docker container. And sets a few environment variables.

__a. The environment variables are used by the datagen.py python program which is the entrypoint into this docker container. This python program is in the project directory and mounted into the docker container.

__i. BROKER environment variable is used by the python program to specify what host to publish data to.

__ii. BROKER_PORT environment variable is used by the python program to specify the port to publish to.

__iii. BROKER_TOPIC environment variable is used by the python program to specify a topic for the publishing of data.

2.5 MESSAGE-BROKER SERVICE

__35. The next service here is a an MQTT message broker. This is the eclipse-mosquitto broker. There isn't too much to it, we use the basic docker image and don't have to do any configuration or pass any options into the docker container when we run it.

```
45  message-broker:
46      image: "eclipse-mosquitto"
47      depends_on:
48          - informix-server
49      container_name: "msgbroker"
50      hostname: "msgbroker"
```

2.6 DATA-READER SERVICE

__36. The next service here is another basic python docker image. This docker container will run a python program that is supplied externally from a mounted directory. This program will subscribe to an **MQTT message broker** retrieve that data and insert it into the **Informix** database.

```
30  data-reader:
31      image: "python:3.6-alpine"
32      depends_on:
33          - message-broker
34      container_name: "datareader"
35      hostname: "datareader"
36      volumes:
37          - ${PWD}/data-reader:/usr/src/myapp
38      working_dir: /usr/src/myapp
39      environment:
40          BROKER: 'msgbroker'
41          BROKER_PORT: 1883
42          BROKER_TOPIC: 'datagen/datagen1'
43          REST_ENDPOINT: 'http://informix:27018/tsdb/tstab_v'
44      entrypoint: python datareader.py
```

__37. This service specifies the container name, hostname and mounted directory as do the others. And specifies environment variables that the python program uses.

__a. BROKER environment variable is used by the python program to specify what host to publish data to.

__b. BROKER_PORT environment variable is used by the python program to specify the port to publish to.

__c. BROKER_TOPIC environment variable is used by the python program to specify a topic for the publishing of data.

__d. REST_ENDPOINT environment variable is used by the python program to specify where to insert the data. (Informix Database)

2.7 GRAFANA-UI SERVICE

__38. The last service uses the base grafana docker image to graph the data from the Informix database server.

```
51  grafana-ui:
52      image: "grafana/grafana"
53      depends_on:
54        - informix-server
55      ports:
56        - 3000:3000
57      container_name: "grafana"
58      hostname: "grafana"
59      volumes:
60        - ${PWD}/grafana/plugins:/var/lib/grafana/plugins
61        - ${PWD}/grafana/provisioning:/etc/grafana/provisioning
62
```

__39. The service specifies the container name, hostname, ports to expose for the UI website. And a couple mounted directories into the docker container.

__a. These are mounted directories from the project and contain the following:

__i. Informix grafana plugin (open source)
https://github.com/OpenInformix/informix-grafana

__ii. Datasource created using the Informix Grafana plugin and a Dashboard to visualize the data.

2.8 RUN SINGLE SERVICE

__40. For the purpose of this lab we will not run a single service, we will run the entire application. But if the need ever arises to run just a single service within a docker-compose.yml file that can be done using the docker-compose up command.

__a. Here we specify the name of the service to be run informix-server.

```
> docker-compose up informix-server
```

3 LAB 3 RUN MICROSERVICES APPLICATION

This lab will run the full microservices application. All components work together to create an entire end to end solution using the building blocks of smaller components.

Docker-compose is a tool designed to run multiple containers that are designed to run and work together.

3.1 FIRST DOCKER-COMPOSE RUN

__41. From the command line in the project folder run docker-compose to start your microservices application. You will see in the output below as it starts each service.

```
> cd compose-microservices-lab
> docker-compose up
```

```
root@informix-VirtualBox:~/lab-darint/compose-microservices-lab# docker-compose
up
Creating network "composemicroserviceslab_default" with the default driver
Creating informix ...
Creating informix ... done
Creating grafana ...
Creating msgbroker ...
Creating grafana
Creating msgbroker ... done
Creating datareader ...
Creating datareader
```

```
Attaching to datagen, datareader, msgbroker, grafana, informix
datagen             | Collecting paho-mqtt
datagen             |   Downloading https://files.pythonhosted.org/package
s/25/63/db25e62979c2a716a74950c9ed658dce431b5cb01fde29eb6cba9489a904/paho-mqtt-1
.4.0.tar.gz (88kB)
datagen             | Building wheels for collected packages: paho-mqtt
datagen             |   Building wheel for paho-mqtt (setup.py): started
datagen             |   Building wheel for paho-mqtt (setup.py): finished
with status 'done'
datagen             |   Created wheel for paho-mqtt: filename=paho_mqtt-1.
4.0-cp36-none-any.whl size=48331 sha256=79c95051ba09f4f32a77a9d17f80af075ec22d66
a902494e1f98b563333e2bbb
datagen             |   Stored in directory: /root/.cache/pip/wheels/82/e5
/de/d90d0f397648a1b58ffeea1b5742ac8c77f71fd43b550fa5a5
datagen             | Successfully built paho-mqtt
datagen             | Installing collected packages: paho-mqtt
datagen             | Successfully installed paho-mqtt-1.4.0
datareader          | Collecting paho-mqtt
datareader          |   Downloading https://files.pythonhosted.org/package
s/25/63/db25e62979c2a716a74950c9ed658dce431b5cb01fde29eb6cba9489a904/paho-mqtt-1
.4.0.tar.gz (88kB)
datareader          | Building wheels for collected packages: paho-mqtt
datareader          |   Building wheel for paho-mqtt (setup.py): started
datareader          |   Building wheel for paho-mqtt (setup.py): finished
with status 'done'
datareader          |   Created wheel for paho-mqtt: filename=paho_mqtt-1.
4.0-cp36-none-any.whl size=48331 sha256=de8d18efde3961ee082a63453eeeaff6d99b08ec
43aa698910162ae338a5534e
datareader          |   Stored in directory: /root/.cache/pip/wheels/82/e5
```

__42. There will be 5 contiainers that have started. One for each service we mentioned in the previous lab. You can see the docker containers with the docker ps command.

```
> docker ps
```

```
root@informix-VirtualBox:~# docker ps
CONTAINER ID        IMAGE                                 COMMAND                  CREATED             STATUS                    PORTS                     NAMES
4f54a13f765b        python:3.6-alpine                     "python datagen.py"      4 minutes ago       Up 4 minutes                                        datagen
610337684f6a        python:3.6-alpine                     "python datareader.py"   4 minutes ago       Up 4 minutes                                        datareader
82b46eef67f0        eclipse-mosquitto                     "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes              1883/tcp                  msgbroker
d669c2b0a982        grafana/grafana                       "/run.sh"                4 minutes ago       Up 4 minutes              0.0.0.0:3000->3000/tcp    grafana
c90585a3c9fe        ibmcom/informix-developer-database    "/opt/ibm/scripts/di…"   4 minutes ago       Up 4 minutes (healthy)    0.0.0.0:27018->27018/tcp  informix
root@informix-VirtualBox:~#
```

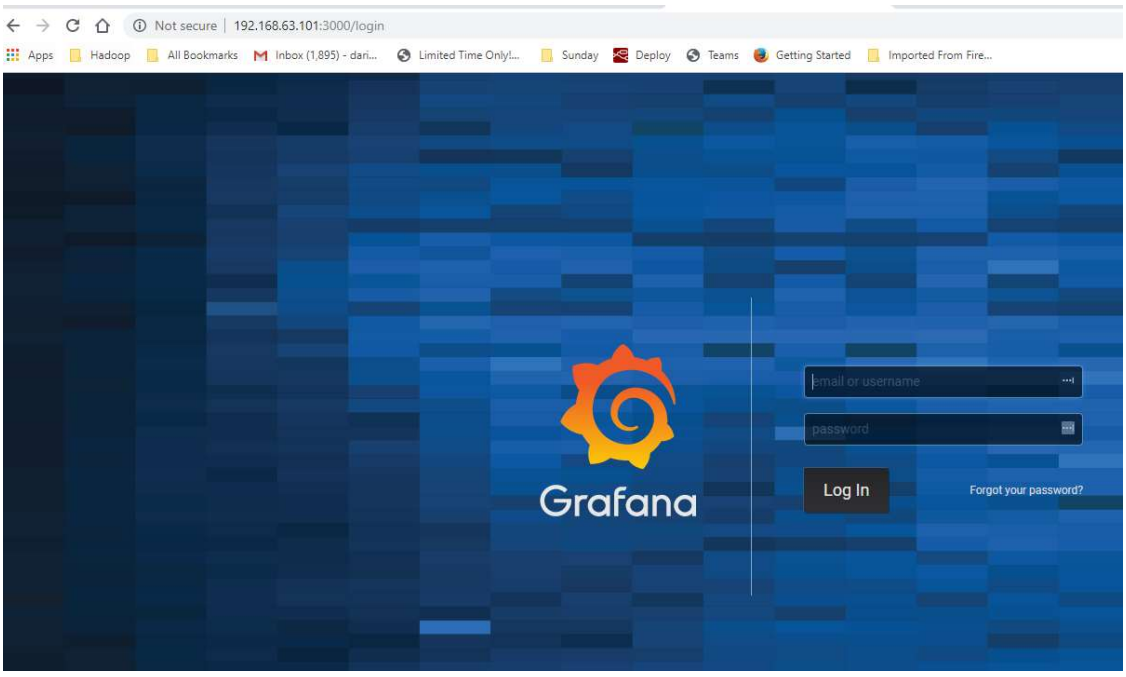
__43. Determine the IP address for the Virtualbox VM. Run ifconfig, look for enp0s8.

```
> ifconfig
```

```
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.63.101  netmask 255.255.255.0  broadcast 192.168.63.255
        inet6 fe80::a00:27ff:fed3:d616  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d3:d6:16  txqueuelen 1000  (Ethernet)
        RX packets 30788  bytes 15451859 (15.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33305  bytes 49337186 (49.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

__a. We see here that the IP address is 192.168.63.101

__44. Direct you browser to http://<ipaddress>:3000

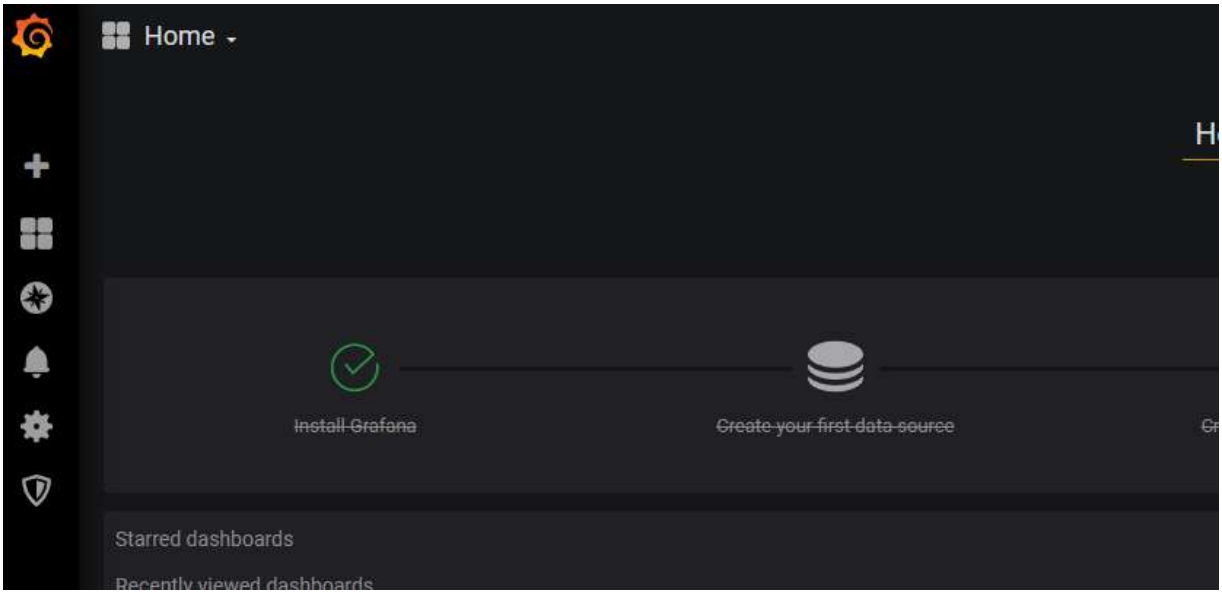


__a. Login to grafana

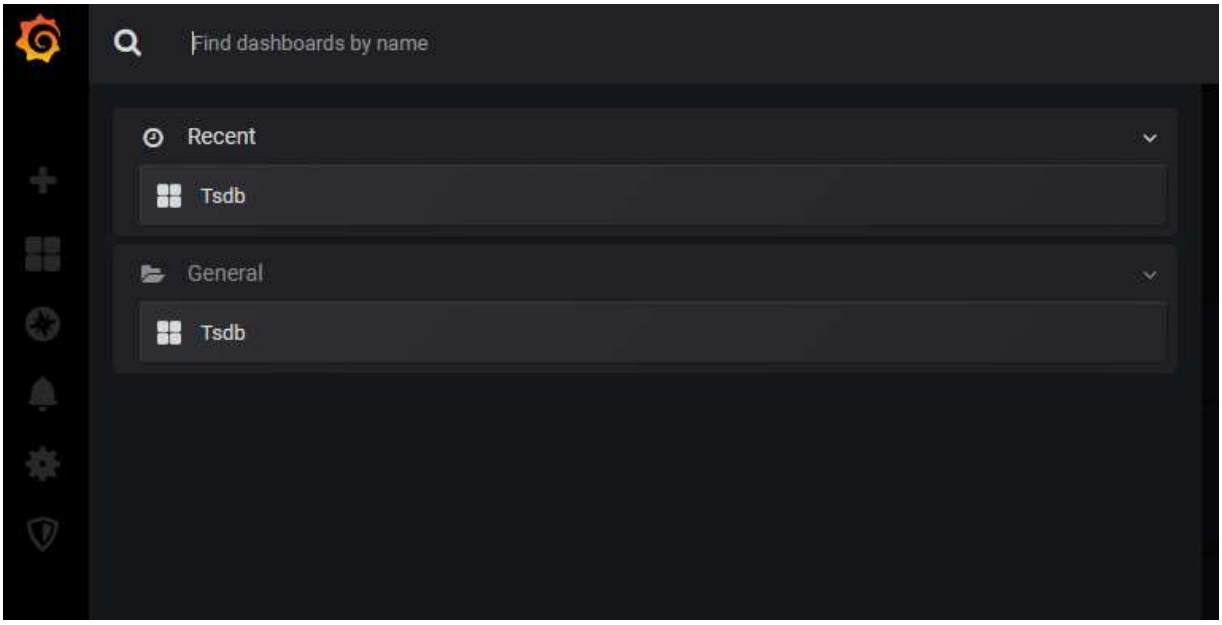
__i. User: admin
__ii. Password: admin

__b. If asked to change the password, simply change to admin123

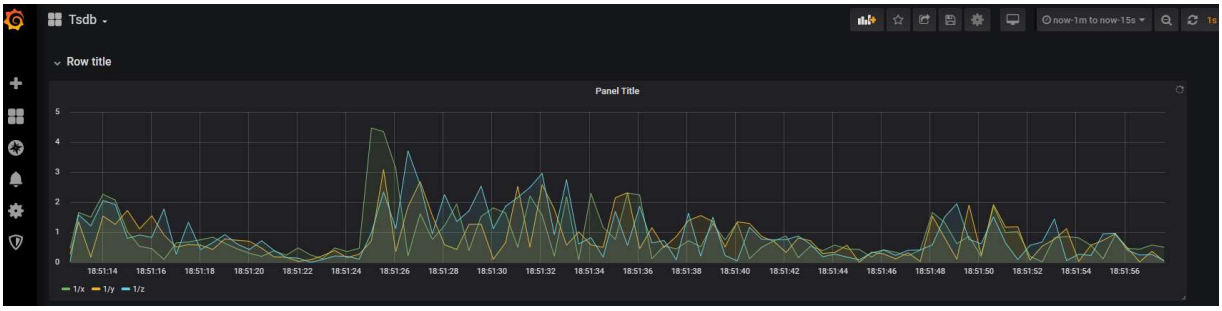
__45. Once logged into Grafana. Click on the Home button in the top left corner.



__46. Then you will see the dasboards that are created. Click on Tsdb under General.



__47. The dashboard will come up and you will see the data visualization.



__48. Here is a big picture and flow of the microservices application and each component (container)

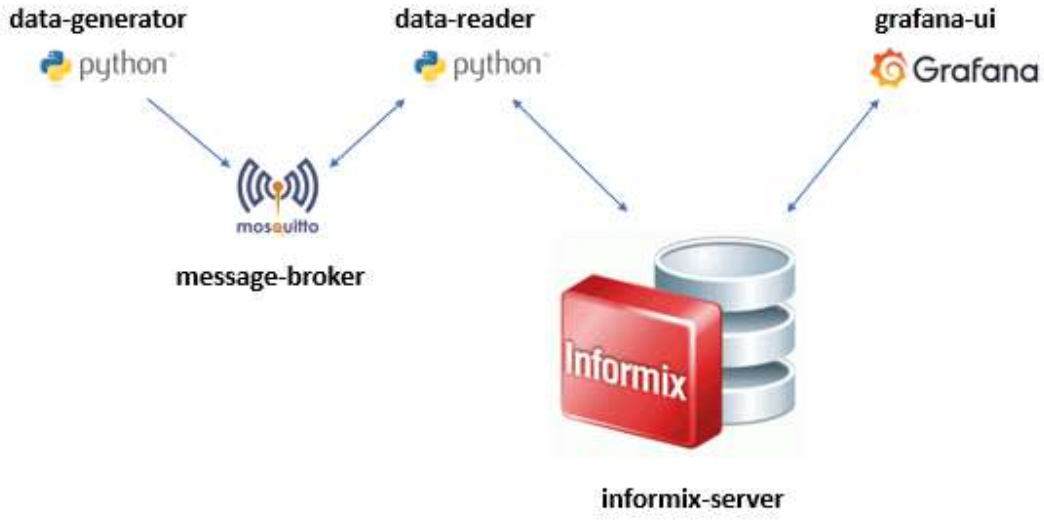
__a. Python container generating data pushing data to a message broker.

__b. Message broker container receiving data from the data generator.

__c. Python container reading data from the message broker and inserting into Informix

__d. Informix Container storing data

__e. Grafana container reading data from Informix and visualizing.



APPENDIX A – DOCKER COMMANDS

docker images

List the docker images that exist in the docker engine

docker ps

List the running docker containers

docker ps -a

List all docker containers (including stopped)

docker start

Start a docker container

docker stop

Stop a docker container

docker exec -it <name> bash

Attach a shell to a running container

docker rm

Remove a container from the docker engine

docker rmi

Remove a docker image

docker volume ls

List volumes

docker volume prune

Delete volume data

docker create volume

Create a named volume

docker commit

Commit a running container to a docker image

docker save

Save a docker image to a tar file

docker load

Load a tar file into the docker engine

docker network ls

List the docker networks available

docker network rm

Remove a docker network

See docker –help for all options.

APPENDIX B – DOCKER-COMPOSE COMMANDS

docker-compose up

Run inside the project directory to start up the entire application.

docker-compose up <service>

Run inside the project directory to start up one or more services that are part of the project.

docker-compose down -v

Run inside the project directory to stop the entire application.

docker-compose build

Command to build a new container with appropriate changes within the project.

docker-compose logs

Command to show the logged data.

See docker-compose –help for all options.