ACIT 3855 – Lab 7 – Containerization

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	End of Next Class: • March 4th for Set C • March 7th for Sets A and B

Purpose

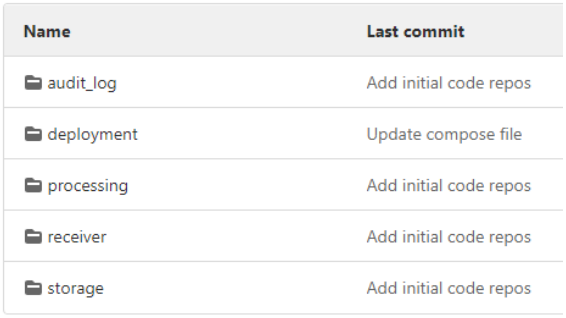
- Use Docker to Containerize our Microservices into self-contained images
- Use Docker Compose to deploy our Microservices

Part 1 – Add Code to Source Code Management

You can use a private (preferred) or public GitHub or GitLab repo(s) for your code.

Create one Git repository for all your services, but put each in a separate folder. The structure should look something like the following:

```
/audit_log/app.py
/audit_log/app_conf.yml
/audit_log/log_conf.yml
/audit_log/openapi.yml
/deployment/docker-compose.yml
/processing/app.py
/processing/app_conf.yml
/processing/log_conf.yml
/processing/openapi.yml
/receiver/app.py
/receiver/app_conf.yml
/receiver/log_conf.yml
/receiver/openapi.yml
/storage/app.py
/storage/app_conf.yml
/storage/log_conf.yml
/storage/openapi.yml
/storage/ - Also your SQLAlchemy object files
and create/drop tables scripts
```



Be careful not to check-in any passwords in your config files. Leave those values blank for now – we will deal with this in a later lab. *I believe one student may have had their MySQL database "hacked" in a past instance of this course because either their passwords were checked into a public GitHub repo or they used a default or simple password!*

Part 2 – Requirements and Dockerfile

For each of your Python based microservices:

Add a **requirements.txt** file that includes the name and version of each 3rd party Python package required for that service (i.e., those you had to explicitly install).

For the Receiver service you need the following in your requirements.txt file:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
```

```
requests==2.25.1
pykafka==2.4.0
```

For the Storage service:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
SQLAlchemy==1.3.22
mysql-connector-python==8.0.23
pymysql==1.0.2
pykafka==2.4.0
```

For the Processing service:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
APScheduler==3.6.3
```

For the Audit Log service:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
requests==2.25.1
pykafka==2.4.0
```

Create a **Dockerfile** that is configured as follows:

- Using a Ubuntu base image
- Installs Python and Pip on top of the base image
- Installs the dependencies from your requirements.txt file
- Runs your Connexion application (i.e., app.py) when the image is run

Here is an example:

```
FROM ubuntu:18.04

LABEL maintainer="mmulder10@bcit.ca"

RUN apt-get update -y && \
    apt-get install -y python3 python3-pip

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip3 install -r requirements.txt

COPY . /app

ENTRYPOINT [ "python3" ]
```

```
CMD [ "app.py" ]
```

Update the maintainer label to be your e-mail address.

Check these files into your Git repositories. Your repositories should look something like this:

```
/audit_log/app.py
/audit_log/app_conf.yml
/audit_log/log_conf.yml
/audit_log/openapi.yml
/audit_log/dockerfile
/audit_log/requirements.txt
/deployment/docker-compose.yml
/processing/app.py
/processing/app_conf.yml
/processing/log_conf.yml
/processing/openapi.yml
/processing/dockerfile
/processing/requirements.txt
/receiver/app.py
/receiver/app_conf.yml
/receiver/log_conf.yml
/receiver/openapi.yml
/receiver/dockerfile
/storage/requirements.txt
/storage/app.py
/storage/app_conf.yml
/storage/log_conf.yml
/storage/openapi.yml
/storage/dockerfile
/storage/requirements.txt
/storage/ - Also your SQLAlchemy object files and create/drop tables scripts
```

Part 3 – Building and Running Your Docker Images

Run your Kafka, Zookeeper and MySQL via Docker Compose:

cd into the deployment folder

```
docker-compose up -d
```

Build and run your Receiver service:

cd into the repo folder for the Receiver service

```
docker build –t receiver:latest .
```

Note: You should see your new image in the list of images via the command “docker images”. The REPOSITORY will be receiver and the TAG latest. It will be assigned an IMAGE ID. You can delete an image via the command “docker image rm <IMAGE ID>”.

Now run the image as a container in the background:

```
docker run -d -p 8080:8080 receiver
```

Note: The -d option runs the container in the background. The -p 8080:8080 option exposes port 8080 on the container for our service.

Repeat for the other services in each of their repo folders:

```
docker build -t storage:latest .

docker run -d -p 8090:8090 storage

docker build -t processing:latest .

docker run -d -p 8100:8100 --network="host" processing
```

Note: We need --network="host" to allow the Processing services to make a RESTful API call to the Storage service on localhost (i.e., http://localhost:8090). The other services are communicating via the Kafka message broker using the host DNS so don't need this.

```
docker build -t processing:latest .

docker run -d -p 8110:8110 audit_log
```

Note: Make sure your app_conf.yml files have valid values, especially for the MySQL DB username and password and Kafka. Since we are running on a single VM, we can still use localhost for the hostname.

Make sure all your services are running by running "docker ps". It should look something like this:

```
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS
06a6a972e9f1        processing                 "python3 app.py"         14 seconds ago      Up 13 seconds
ygin
cfea2f8c4187        audit_log                  "python3 app.py"         About a minute ago  Up About a minute   0.0.0.0:8110->8110/tcp
slett
5f9bef075255        storage                    "python3 app.py"         12 hours ago        Up 12 hours         0.0.0.0:8090->8090/tcp
ds
e6bc531cf7f9        receiver                   "python3 app.py"         12 hours ago        Up 12 hours         0.0.0.0:8080->8080/tcp
well
0c1bf115253f        wurstmeister/kafka         "start-kafka.sh"         6 days ago          Up 12 hours         0.0.0.0:9092->9092/tcp
fka_1
d3374aca90b9        mysql:5.7                  "docker-entrypoint.s…"   6 days ago          Up 13 hours         0.0.0.0:3306->3306/tcp, 33060/tcp
_1
2dbcf43ea6a7        wurstmeister/zookeeper     "/bin/sh -c '/usr/sb…"   6 days ago          Up 12 hours         22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:32768->2181/tcp
okeeper_1
```

Make sure each service has not errors while running. Use docker logs <container id> to see the log output for each service.

Make sure to commit your requirements.txt and Dockerfile to your GitHub/GitLab repositories.

Part 4 – Testing your Services

On your Cloud VM, open up the following ports to allow access to your services with externally available endpoints:

- Receiver – Port 8080
- Processing – Port 8100
- Audit Log – Port 8110

Using PostMan, exercise the following endpoints on the services running on your Cloud VM. You will need to use the DNS name of your VM rather than localhost for the URLs:

- Receiver – POST an event of each type
- Processing – GET the stats
- Audit Log – GET the first event of each type

Once you are convinced they all run, stop each container before moving on to Step 5.

Part 5 – Configure and Run our Docker Images with Docker Compose

You are going to add your services to the deployment/docker_compose.yml file of your repo so you will no longer have to run each service individually.

Make sure the repos for your services and the Docker images are up-to-date on your Cloud VM (this should be the case if you finished the above steps).

Configure your microservices in docker_compose.yml as follows:

- Add a new service for each of your microservices naming it appropriately (i.e., receiver, storage, processor, audit_log)
- Set the image and ports properties for each microservice. The image is the name of the Docker image you built for a particular service (run *docker images* to see all the locally built images) and the ports should correspond to the port on which the Flask server is running for that service.
- Set the depends_on property for the microservice to kafka and/or db if it requires either or both of those services to be up and running. The depends_on property should also be set if a service depends on another service (i.e., Processing depends on Storage since it calls the GET endpoints on Storage)
- Here is a summary of what you configuration for each service should be:

receiver	ports: 8080:8080 depends on kafka (since it's a producer)
storage	ports: 8090:8090 depends on kafka (since it's a consumer) and db
processing	ports: 8100:8100 network mode is host (i.e., network_mode: "host"). It needs to access the storage service endpoint (on localhost) to get new events. depends on storage
audit_log	ports: 8110:8110 depends on kafka (since it's a consumer)

Here is an example for the receiver:

```
receiver:
  image: receiver
  ports:
    - "8080:8080"
  depends_on:
    - "kafka"
```

Make sure all your services are stopped as well as zookeeper, kafka and mysql (docker-compose down).

Now bring all the services up as follows with Docker Compose:

docker-compose up –d (note the –d runs the services in the background)

Test your services running on your VM with Postman to make sure they still all work. Troubleshoot as necessary.

Also test with jMeter. Make sure to update the hostname of the receiver service in your test case. You will be asked to demostrate that your services are correctly running on your Cloud VM by running your jMeter load test against them.

Grading and Submission

Submit the following to the Lab 7 Dropbox on D2L:

- Your docker-compose.yml file

Demonstrate the following to your instructor:

Demonstrate that you can run each service individually with a docker run command and view the console or log output.	4 marks
Demonstrate that you can run all your services with Docker Compose. Exercise the following: • Your jMeter test against those services • GET of your /stats endpoint • GET of one or both of your Audit log endpoints	6 marks