ACIT 3855 Lab 7 - Deployment and Reverse Proxy

Part 1 - Setting up Nginx Reverse Proxy

Begin by ensuring your VM has Nginx installed:

```
sudo apt update
sudo apt install nginx
```

Note that if your VM already has Apache installed and running from a previous course/project, you will need to stop the Apache service. How to do this depends on your VM's operating system and configuration, but a few possibilities include:

```
sudo /etc/init.d/apache2 stop
# and/or
sudo service apache2 stop
# and/or
sudo systemctl stop apache2.service
```

and you can check to see if any processes are using port 80 by running:

```
lsof -i :80
```

Once you have verified Nginx is running via the following command:

```
sudo /etc/init.d/nginx status
```

and confirm that visiting your IP/url in the browser results in seeing the default 'Welcome to nginx!' page is visible, you will need to make changes to the default configuration so that we can:

1. Show the dashboard by default instead of the welcome page
2. Use pretty URLs to access our services, e.g. `/receiver/ui` or `/storage/ui`

Update Nginx's default at /etc/nginx/sites-available/default replacing the existing location block:

```
location / {
    try_files $uri $uri/ =404;
}
```

with the following:

```
location / {
    proxy_pass http://localhost:3000;
}

location /receiver {
    proxy_pass http://localhost:8080;
}

location /storage {
    proxy_pass http://localhost:8090;
}

location /processing {
    proxy_pass http://localhost:8100;
}
```

This ensures that requests to the above base URLs will be forwarded to our services. After making the above changes, run:

```
nginx -t
```

to make sure your syntax is correct, followed by:

```
/etc/init.d/nginx restart
# or systemctl restart nginx, etc.
```

Part 2 - Installing Docker

Begin by ensuring `curl` is installed in your VM by running `curl --version`. If you do not get version information back, it can be installed on Ubuntu via:

```
sudo apt update
sudo apt upgrade
sudo apt install curl
```

Once curl is installed, run the following commands on Ubuntu to install docker and docker.io:

```
apt install docker.io
mkdir -p ~/.docker/cli-plugins/
curl -SL \
https://github.com/docker/compose/releases/download/v2.3.3/docker-compose-linux-x86_64 \
-o ~/.docker/cli-plugins/docker-compose
chmod +x ~/.docker/cli-plugins/docker-compose
```

If all has gone well you should now be able to run:

```
docker compose version
```

and see some output.

With docker installed you can now migrate your containers from the previous lab over to your VM (via git or any other means) and update them.

Part 3 - Updating and running your Dockerized services

Start by bringing up Kafka, Zookeeper, and MySQL using the following docker-compose file:

```
version: '2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - 22181:2181
  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 29092:29092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  mysql:
    container_name: mysql
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - data:/var/lib/mysql
volumes:
  data:
```

and running:

```
docker compose up -d
```

IMPORTANT NOTE: you will need to recreate the database and tables (as well as possibly changing the authentication plugin for 'root'@'localhost' and/or 'root'@'%' in your MySQL container before continuing, otherwise your services may not run)

Before building and running each of your services on the VM, you will need to make one change so that Nginx correctly resolves requests. In each service's `app.py`, add the base*path argument to your existing app.add*api() as follows:

```
app.add_api("openapi.yml", base_path="/receiver",
    strict_validation=True, validate_responses=True)
```

Note: change `/receiver` above to the appropriate name for your service.

Next, build each service as you did in the previous lab from within each service's folder, e.g.:

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

and create a container ensuring it is attached to the `kafka_default` network, e.g.:

```
docker run -d -p 8080:8080 --network=kafka_default --name=receiver receiver
```

Finally, ensure that all services are up and running with `docker ps`

Part 4 - Deploying the dashboard

The dashboard app does not need to be dockerized – we can run it on the server directly using node.js and a process manager to ensure that it restarts automatically whenever the VM is restarted.

Begin by installing node.js and npm on the VM:

```
sudo apt install nodejs npm
```

Once node and NPM are installed, globally install the PM2 process manager:

```
npm install -g pm2
```

Once your dashboard app has been cloned/moved to your VM, run

```
npm install
```

To ensure that all dependencies are installed, then

```
pm2 start npm --name "dashboard" -- start
```

to allow PM2 to manage restarting the app when necessary. Verify that everything is configured correctly by visiting the root IP/URL of your VM. With a hard-refresh of your cache, you should now see your dashboard UI running instead of the default Nginx welcome page.

Testing and Submission

If configured correctly, you should now be able to view the dashboard at your root url, and any /ui pages etc. that your services expose.

More importantly, you must verify that you can now send a POST request to your VM's receiver address, e.g. https://my-horrifically-complicated-aws-domain/receiver/buy and the data is stored in your MySQL database.

Submit a .txt file to the Lab 7 folder with the URL/IP of your VM, as well as a screenshot showing the output of Docker Compose with all services successfully up and running.