ACIT 3855 – Lab 11 – Setting Up a Load Balancer

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	Before the end of last class: • April 8th for Set C • April 11th for Sets A and B

Purpose

- Scale the Receiver and Storage Services using a software load balancer (NGINX)
- Move all the services to a common endpoint
- Verify your stats are correct after running with jMeter

Part 1 – Add NGINX Configuration and Context Paths

On your Cloud VM, add an nginx folder to your home directory (i.e., /home/<username>/nginx)

Add a file called nginx.conf in that folder with the following content:

```
user nginx;
# can handle 1000 concurrent connections
events {
    worker_connections   1000;
}
# forwards http requests
http {
        # http server
        server {
              # listens the requests coming on port 80
              listen 80;
              access_log  off;
              # / means all requests on /dashboard will be forwarded to the dashboard ui service
              location / {
                # resolves the IP of dashboard using Docker internal DNS
                proxy_pass http://dashboard-ui:3000;
              }
              # / means all the requests on /receiver will be forwarded to receiver service
              location /receiver {
                # resolves the IP of receiver using Docker internal DNS
                proxy_pass http://receiver:8080;
              }
              # / means all the requests on /storage will be forwarded to storage service
              location /storage {
                 # resolves the IP of storage using Docker internal DNS
                proxy_pass http://storage:8090;
              }
              # / means all the requests on /processing will be forwarded to processing service
              location /processing {
                # resolves the IP of processing using Docker internal DNS
                proxy_pass http://processing:8100;
              }
              # / means all the requests on /audit_log will be forwarded to audit_log service
              location /audit_log {
                # resolves the IP of audit_log using Docker internal DNS
                proxy_pass http://audit_log:8110;
              }
        }
}
```

Update the following line in the app.py file of each of your services:

Current:
```
app.add_api("openapi.yml", strict_validation=True, validate_responses=True)
```

Change the Receiver service to:
```
app.add_api("openapi.yml", base_path="/receiver", strict_validation=True,
validate_responses=True)
```

Change the Storage service to:
```
app.add_api("openapi.yml", base_path="/storage", strict_validation=True, validate_responses=True)
```

Change the Processing service to:
```
app.add_api("openapi.yml", base_path="/processing", strict_validation=True,
validate_responses=True)
```

Change the Audit_Log service to:
```
app.add_api("openapi.yml", base_path="/audit_log", strict_validation=True,
validate_responses=True)
```

This sets the context path of each service so the base endpoint will be:
http://<Cloud VM DNS>:<Service Port>/<Service Context>

For the Processing and Audit_Log services, disable the CORS in the test environment. Update the app.py for each as follows:

```
if "TARGET_ENV" not in os.environ or os.environ["TARGET_ENV"] != "test":
    CORS(app.app)
    app.app.config['CORS_HEADERS'] = 'Content-Type'
```

CORS is not needed when all services are on the same domain and port, which we will setup in Part 2.

Rebuild Docker image for each service.

Part 2 – Add NGINX and Service Updates in Docker Compose

We will be adding a Docker network on which NGINX can forward requests to each of the services RESTful API.

Add the following to the end of your docker-compose.yml file to add this network:

```
networks:
  api.network:
```

This will create network on which the participating services can communicate.

Modify the following for each of the receiver, storage, processing and audit_log services in the docker-compose.yml file. This will connect each service to the above network. We are also changing the port from 8080:8080 to just 8080. This will assign a random port to each instance

of that container but map it to port 8080 (for the receiver service) in the running application in the container.

Receiver:

```
  receiver:
    image: receiver
    ports:
      - "8080"
    networks:
      - "api.network"
```

Storage:

```
  storage:
    image: storage
    ports:
      - "8090"
    networks:
      - "api.network"
```

Processing:

```
  processing:
    image: processing
    ports:
      - "8100"
    networks:
      - "api.network"
```

Make sure to remove the host networking option for the processing service.

Audit_Log:

```
  audit_log:
    image: audit_log
    ports:
      - "8110"
    networks:
      - "api.network"
```

Dashboard:

```
  Dashboard-ui:
    image: dashboard-ui
    ports:
      - "3000"
    networks:
      - "api.network"
```

Add NGINX as a service in your docker-compose.yml file:

```
 nginx:
    image: nginx:latest
    # Connects the conf file of the container to the conf file in our folder
    volumes:
      - /<Your Home Directory>/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    # It will start up the nginx only when all api containers have started
    depends_on:
      - "receiver"
      - "storage"
      - "processing"
      - "audit_log"
```

```
  - "dashboard-ui"
# Connects the port 80 of the nginx container to localhost:80 or localhost
ports:
  - "80:80"
networks:
  - "api.network"
```

The NGINX service depends on each of our four services. It will listen on Port 80 (the default http port) and will redirect requests to the appropriate service based on the context path. For instance:

Requests on http://<Cloud VM DNS>/receiver are directed to the Receiver service
Requests on http://<Cloud VM DNS>/storage are directed to the Storage service
Requests on http://<Cloud VM DNS>/processing are directed to the Processing service
Requests on http://<Cloud VM DNS>/audit_log are directed to the Audit_Log service
Requests on http://<Cloud VM DNS> are directed to the Dashboard service

Update the app_conf.yml file of the Processing service (in the /home/<username>/config/processing folder) with the new URL endpoint for the Storage Service:

http://<Cloud VM DNS>/storage

The Processing Service will now use the external URL on port 80 will go through the NGINX load balancer as we will be scaling the Storage service.

On your Cloud VM, update your Networking rules as follows:

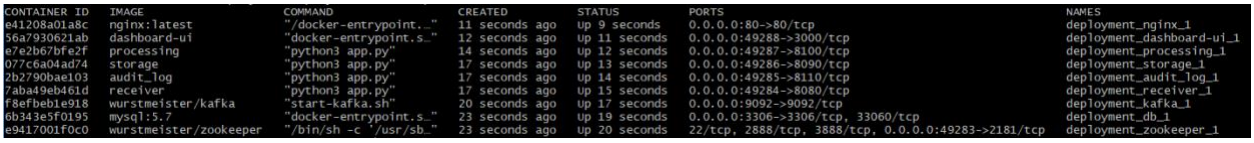
- Remove any rules allowing inbound requests on Ports 3000, 8080, 8090, 8100 and 8110.
- Add an inbound rule allowing access on Port 80

Part 3 – Scaling and Testing

Using Docker Compose, bring up your services and make sure they all run:

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

The output from docker ps should look something like:



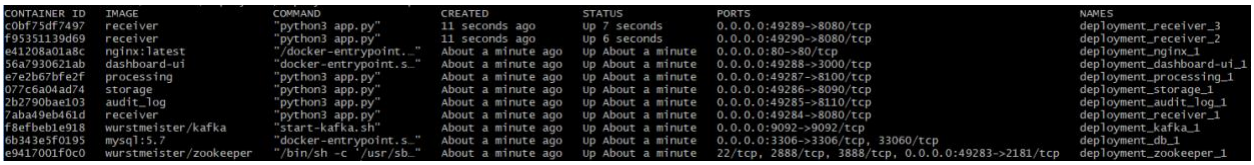
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e41208a01a8c	nginx:latest	"/docker-entrypoint..."	11 seconds ago	Up 9 seconds	0.0.0.0:80->80/tcp	deployment_nginx_1
56a7930621ab	dashboard-ui	"docker-entrypoint.s..."	12 seconds ago	Up 11 seconds	0.0.0.0:49288->3000/tcp	deployment_dashboard-ui_1
e7e2b67bfe2f	processing	"python3 app.py"	14 seconds ago	Up 12 seconds	0.0.0.0:49287->8100/tcp	deployment_processing_1
077c6a04ad74	storage	"python3 app.py"	17 seconds ago	Up 13 seconds	0.0.0.0:49286->8090/tcp	deployment_storage_1
2b2790bae103	audit_log	"python3 app.py"	17 seconds ago	Up 14 seconds	0.0.0.0:49285->8110/tcp	deployment_audit_log_1
7aba49eb461d	receiver	"python3 app.py"	17 seconds ago	Up 15 seconds	0.0.0.0:49284->8080/tcp	deployment_receiver_1
f8efbeb1e918	wurstmeister/kafka	"start-kafka.sh"	20 seconds ago	Up 17 seconds	0.0.0.0:9092->9092/tcp	deployment_kafka_1
6b343e5f0195	mysql:5.7	"docker-entrypoint.s..."	23 seconds ago	Up 19 seconds	0.0.0.0:3306->3306/tcp, 33060/tcp	deployment_db_1
e9417001f0c0	wurstmeister/zookeeper	"/bin/sh -c '/usr/sb..."	23 seconds ago	Up 20 seconds	22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:49283->2181/tcp	deployment_zookeeper_1

Bring your services down using “docker-compose down”

Bring your services up again but with scaled receiver and storage services:

```
docker-compose up -d --scale receiver=3
```

Running docker ps should now look something like (i.e., 3 receiver containers running):

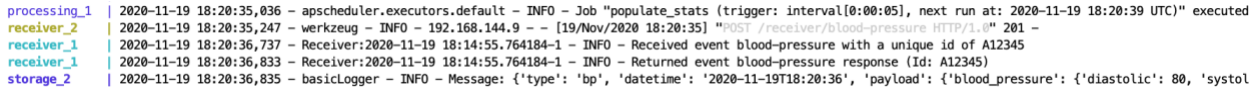


```
CONTAINER ID   IMAGE                      COMMAND                  CREATED              STATUS              PORTS                                            NAMES
c0bf75df7497   receiver                   "python3 app.py"         11 seconds ago       Up 7 seconds        0.0.0.0:49289->8080/tcp                          deployment_receiver_3
f95351139d69   receiver                   "python3 app.py"         11 seconds ago       Up 6 seconds        0.0.0.0:49290->8080/tcp                          deployment_receiver_2
e41208a01a8c   nginx:latest               "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp                               deployment_nginx_1
56a7930621ab   dashboard-ui               "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:49288->3000/tcp                          deployment_dashboard-ui_1
e7e2b67bfe2f   processing                 "python3 app.py"         About a minute ago   Up About a minute   0.0.0.0:49287->8100/tcp                          deployment_processing_1
077c6a04ad74   storage                    "python3 app.py"         About a minute ago   Up About a minute   0.0.0.0:49286->8090/tcp                          deployment_storage_1
2b2790bae103   audit_log                  "python3 app.py"         About a minute ago   Up About a minute   0.0.0.0:49285->8110/tcp                          deployment_audit_log_1
7aba49eb461d   receiver                   "python3 app.py"         About a minute ago   Up About a minute   0.0.0.0:49284->8080/tcp                          deployment_receiver_1
f8efbeb1e918   wurstmeister/kafka         "start-kafka.sh"         About a minute ago   Up About a minute   0.0.0.0:9092->9092/tcp                           deployment_kafka_1
6b343e5f0195   mysql:5.7                  "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:3306->3306/tcp, 33060/tcp                deployment_db_1
e9417001f0c0   wurstmeister/zookeeper     "/bin/sh -c '/usr/sb…"   About a minute ago   Up About a minute   22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:49283->2181/tcp   deployment_zookeeper_1
```

Change your URLs in Postman to correspond to the base URLs:

- http://<Cloud VM>/receiver
- http://<Cloud VM>/processing
- http://<Cloud VM>/audit_log

If you run “docker-compose logs –f” and use Postman to send in events to the Receiver service, you should see different receiver containers handling each. Note that each container name is color-coded in the logs to highlight the differences.



```
processing_1  | 2020-11-19 18:20:35,036 - apscheduler.executors.default - INFO - Job "populate_stats (trigger: interval[0:00:05], next run at: 2020-11-19 18:20:39 UTC)" executed
receiver_2    | 2020-11-19 18:20:35,247 - werkzeug - INFO - 192.168.144.9 - - [19/Nov/2020 18:20:35] "POST /receiver/blood-pressure HTTP/1.0" 201 -
receiver_1    | 2020-11-19 18:20:36,737 - Receiver:2020-11-19 18:14:55.764184-1 - INFO - Received event blood-pressure with a unique id of A12345
receiver_1    | 2020-11-19 18:20:36,833 - Receiver:2020-11-19 18:14:55.764184-1 - INFO - Returned event blood-pressure response (Id: A12345)
storage_2     | 2020-11-19 18:20:36,835 - basicLogger - INFO - Message: {'type': 'bp', 'datetime': '2020-11-19T18:20:36', 'payload': {'blood_pressure': {'diastolic': 80, 'systol
```

Make a screenshot of this for your submission.

Grading and Submission

Submit the following to the Lab 11 Dropbox on D2L:

- Your updated docker-compose.yml
- The screenshot of the docker-compose logs showing that your load balancer is working.

Show that you have updated your system with the Nginx Load Balancer: • docker-compose.yml is updated • Single endpoint for all services	6 marks
Demo with jMeter • Minimum of 100 concurrent threads • Updated to the new endpoints • Processing counts are exactly as expected	4 marks
Total	**10 marks**