ACIT 3855 – Lab 10 – Externalized Configs, Logging and Volumes

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	Demo by end of class on: • April 8th (or earlier) for Set C • April 4th for Sets A and B

Purpose

- Externalize your service configurations to a directory on the Cloud VM
- Aggregate your log files to a directory on the Cloud VM
- Persist the Processing Service's stats.sqlite file to a Docker Volume

Part 1 – Externalize Configurations

We are going to have our services use the app_conf.yml and log_conf.yml files from a directory on the Cloud VM rather than the one built into the image.

Create a directory called config in the home directory of your Cloud VM:

/home/<username>/config

Create a subdirectory for each service:

/home/<username>/config/receiver
/home/<username>/config/storage
/home/<username>/config/processing
/home/<username>/config/audit_log

Copy the app_conf.yml and log_conf.yml files from the service repositories to the correct folder above. Each folder should look like this:

/home/<username>/config/receiver/app_conf.yml
/home/<username>/config/receiver/log_conf.yml

Now edit your docker-compose.yml file as follow to mount the config directory into each of your services:

```
services:
   receiver:
     …
     environment:
        - TARGET_ENV=test
```

```
volumes:

  - /home/<username>/config/receiver:/config

depends_on:

  - "kafka"
```

This does the following:

- Creates an environment variable TARGET_ENV set to the string test
- Creates a bind mount, mapping the /config directory inside the container to the /home/<username>/config/receiver directory on the Cloud VM.

Now modify the code that loads in the application and log configurations in the app.py of each of your services:

```
import os

if "TARGET_ENV" in os.environ and os.environ["TARGET_ENV"] == "test":
    print("In Test Environment")
    app_conf_file = "/config/app_conf.yml"
    log_conf_file = "/config/log_conf.yml"
else:
    print("In Dev Environment")
    app_conf_file = "app_conf.yml"
    log_conf_file = "log_conf.yml"

with open(app_conf_file, 'r') as f:
    app_config = yaml.safe_load(f.read())

# External Logging Configuration
with open(log_conf_file, 'r') as f:
    log_config = yaml.safe_load(f.read())
    logging.config.dictConfig(log_config)

logger = logging.getLogger('basicLogger')

logger.info("App Conf File: %s" % app_conf_file)
logger.info("Log Conf File: %s" % log_conf_file)
```

Now make sure all your running containers are stopped and removed:

```
docker-compose down
docker ps should now show no containers running
docker ps –a will show a number of stopped containers
docker container prune will remove all the stopped containers
```

Now build each of your images (Receiver, Storage, Processing, Audit Log):

cd into the service folder from your repo:

```
docker image rm <image name, i.e., receiver>
docker build -t <image name>:latest .
```

You can test your individual images as follows to verify you get the expected configuration file path in the logged output:

docker run -p "<port>:<port>" -e TARGET_ENV=test --mount type=bind,source=/home/<username>/config/storage,target=/config <service image name>

It should log the following to the console if it's using the configuration files in your mounted directory:

```
App Conf File: /config/app_conf.yml

Log Conf File: /config/log_conf.yml
```

Use docker-compose to bring up all your services and make sure they run successfully.

You should now be able to stop a container. Make a change to its config file in /home/<username>/config/<service name>/app_conf.yml. Then restart the container (you can just run docker-compose up –d to do this) to have the change take effect.

Use the docker logs –f <container id> command to follow the logs for a specific running container.

Part 2 – Aggregating Logs

Now that our configurations are externalized, we can change the location where logs are written in log_conf.yml.

First, let's create a new directory on the Cloud VM to aggregrate the log files:

/home/<username>/logs

Mount this directory to each of your services in docker-compose.yml (make sure to use the correct config directory for each service):

```
volumes:
  - /home/<username>/config/<service name>:/config
  - /home/<username>/logs:/logs
```

Modify the log_conf.yml file for each service in the /home/<username/config/<service name/ directories:

```
file:
  class: logging.FileHandler
  level: DEBUG
  formatter: simple
  filename: /logs/app_receiver.log
```

Making the filename unique for each service.

Bring all your services down and then back up with docker-compose. You should see the log files for each service appear in the /logs directory on the Cloud VM. This is our basic log aggregration so we don't have to go into your running container to see the log file.

If you want to connect to your running container, the docker exec command lets your run a linux command on the container. And docker exec bash lets your run a bash shell on the container. I.e., docker exec -it /bin/bash

Part 3 – Docker Volume for Processing Service

When a Docker container is stopped, it does retain the data stored within its filesystem. However, if the container is removed, that data goes away. So normally we store persistent file based data in a Docker volume. That's what we're going to do for the SQLite data store file of the Processing service.

In your docker-compose.yml file, add the following to the specification of the volumes:

```
volumes:
  my-db:
  processing-db:
```

And add the following to the Processing service specification:

```
     volumes:
       - /home/<username>/config/processing:/config
       - /home/<username>/logs:/logs
       - processing-db:/data
```

In the app_conf.yml file for the Processing service in your /home/<username>/config/processing directory, modify the following:

```
datastore:
  filename: /data/stats.sqlite
```

Note: You will need to update your Processing service so it creates the SQLite with an empty stats table. You can try doing this by:

- Having your Processing service check for the existence of the file on startup
- If it does not exist, create the SQLite file with an empty stats table.

Once you've made these changes and re-built your Docker image for the Processing service to get your code changes… then you can bring down all your services with docker-compose down and then bring them back up with docker-compose up -d

Your Processing service should now be storing stats.sqlite in the /data directory corresponding to the processing-db volume.

Verify this in the following two ways:

- Run jMeter to populate your stats.sqlite file. Then connect to the running Processing service container as follows:

  docker exec -it <container id> bash

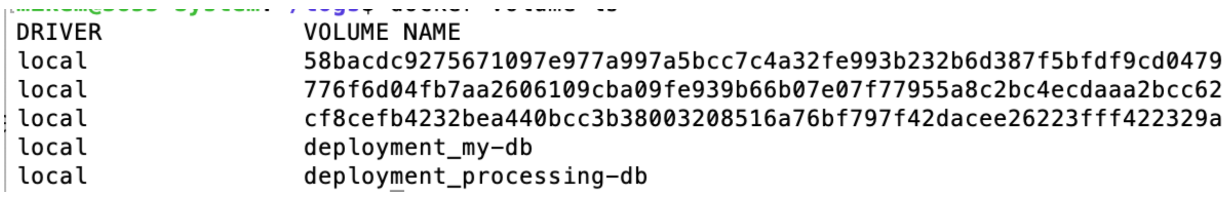
  Use docker ps to get the container id.

  Then navigate to the /data directory and you should see your stats.sqlite file. Check that it contains some values inside.

  Type exit to exit the bash session on the container

- Run jMeter to populate your stats.sqlite file. Then do the following:
  - Check the stats using your /stats endpoint
  - Stop the Processing service container (docker stop <container id)
  - Remove the Processing service container:
    - docker ps -a (lists stopped containers too)
    - docker rm <container id>
  - Run docker-compose up –d to bring the Processing service back up
  - Check that the stats are retained using your /stats endpoint

You can also check that the volume (processing-db) now exists by running docker volume ls



```
DRIVER              VOLUME NAME
local               58bacdc9275671097e977a997a5bcc7c4a32fe993b232b6d387f5bfdf9cd0479
local               776f6d04fb7aa2606109cba09fe939b66b07e07f77955a8c2bc4ecdaaa2bcc62
local               cf8cefb4232bea440bcc3b38003208516a76bf797f42dacee26223fff422329a
local               deployment_my-db
local               deployment_processing-db
```

Grading and Submission

Submit the following to the Lab 10 Dropbox on D2L:

- Your updated docker-compose.yml

Demonstrate that a configuration change in one of your external configurations is applied to the service.	3 marks
Demonstrate that log messages are written to your logs folder for each service.	3 marks
Demonstrate that the stats.sqlite file is retained if you stop/remove the Processing Service container and then re-run it. Also show the docker volume exists using the "docker volume ls" command	4 marks
Total	**10 marks**