Containers 101 Workshop Exercises

Neetha Ravishankar, Imane Qadiri

This workshop will help you create a containerized micro-services application using Docker. You be learning to write Dockerfiles, use basic Docker commands to build images and run containers using the images. You will also get to setup a network for containers to interact with each other and learn how to create a persisting storage for containers. You will write a docker compose file that creates and runs containers as the first step towards learning about Container Orchestration.

Table of Contents

1. Prerequisites

- Laptop running 64-bit operating system with at least 4GB of RAM.
    - Mac with Intel Chip: macOS must be version 10.15 or newer. That is, Catalina, Big Sur, or Monterey.
    - Mac with Apple silicon: Rosetta 2 for best experience.
    - Windows 10 or higher with virtualization enabled.
    - Supported Linux distributions and pre-requisites for each can be found here: https://docs.docker.com/engine/install/ubuntu/
- Install latest docker desktop for Mac and Windows. Ensure latest Docker Engine is installed - not older than version 20.10.5
    - Instructions for Mac - https://docs.docker.com/desktop/mac/install/
    - Instructions for Windows - https://docs.docker.com/desktop/windows/install/
    - Instructions for Linux distributions - https://docs.docker.com/engine/install/
- Sign up for a free Personal Docker Hub account which gives access to a wide range of images from publicly available repositories as well as store images in your own private repository. https://hub.docker.com/signup
- [OPTIONAL] Install aws cli tool.
    - Instructions - https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

2. Login and Verification

Login to Docker Desktop or vis cli using Docker ID and password created during registration of Docker hub account. This will allow us to pull and push images to Docker hub. Use one of the following two options:

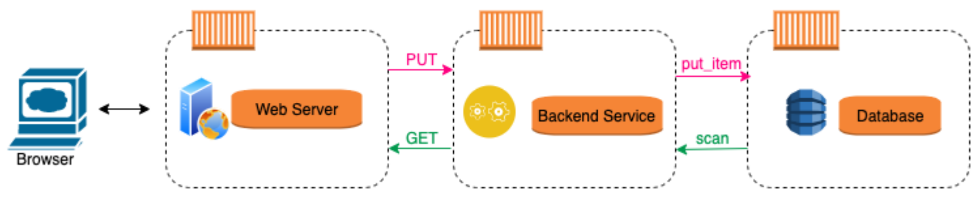
- Login using Docker desktop (only for Mac and Windows). Open Docker desktop application and click on Sign In. Enter ID and password to login.
- Login via Docker cli using following command.

  `docker login`

3. Demo 1 [Optional] - Isolating File System

1. Pull latest Alpine image
   `docker pull alpine`
2. Run alpine container in interactive mode
   `docker run -it alpine`
3. Make a new directory and cd to it
   `mkdir new-fs`
   `cd new-fs`
4. Copy `/bin` and `/lib` directories to new folder
   `cp -r /bin . && cp -r /lib .`
5. Create a dummy file in the directory using vi and save it
6. Remove rm binary
   `rm bin/rm`
7. Change root of the shell to new-fs
   `chroot .`
8. Try deleting the newly created file
   `rm newfile.txt`
   This will fail!
9. Just like that you created a shell with custom behavior
10. Play around with this concept. Install specific libraries and applications, customize your own FS. You can tar the shell and share it which is what a container image is in highly simplifies terms.

Overview of the containerized application you will be building in this workshop



4. Exercise 1 - Creating Dockerfile, building the image and running a container

1. Create and run a python flask server on a container
   a. Open and edit the file - `exercises/backend-service/src/server.py`
   b. Import flask and initialize a flask server
   c. Start the flask server in the main function
2. Add Flask as a required dependency in requirements.txt

`Flask==2.0.3`

3. Open the empty Docker file and add instructions to build an image
    a. Set base image
    `FROM python:3.8-alpine`
    b. Set working directory
    `WORKDIR /src`
    c. Copy required files
    ```
    COPY requirements.txt .
    COPY ./src /src
    ```
    d. Install required dependencies
    `RUN pip install -r requirements.txt`
    e. Expose required port and execute python script
    ```
    EXPOSE 5000
    CMD python server.py
    ```
4. Build the image using docker build command
    a. `docker build . -- tag backend-service:latest`
5. Run a container using the image built
    a. `docker run -it backend-service`

5. Demo 2 - Data Persistence

1. Pull and run alpine container
2. Create a new folder and some files inside the container
3. Exit the container
4. Run the container again. Newly created data is lost! By default containers don’t persist data.

6. Demo 3 [OPTIONAL] - Persisting data in DynamoDB

1. Complete the Dockerfile provided under `exercises/db`
    ```
    FROM amazon/dynamodb-local
    WORKDIR /home/dynamodblocal
    RUN mkdir ./db && chown -R 1000 ./db
    EXPOSE 8080
    CMD ["-jar", "DynamoDBLocal.jar", "-dbPath", "./db", "-sharedDb", "-port", "8080"]
    VOLUME ["./db"]
    ```
2. Build and run the image
    ```
    docker build . -- tag db:latest
    docker run -it -p 8080:8080 db
    ```
3. Use aws cli to try and connect to the dynamodb running locally on the container
    a. List tables to view existing tables
    `aws dynamodb list-tables --endpoint-url localhost:8080`
    b. There are no existing tables found.
    c. Create a new table for testing purposes
    ```
    aws dynamodb create-table --endpoint-url http://localhost:8080 --table-name
    Music --attribute-definitions AttributeName=Artist,AttributeType=S
    AttributeName=SongTitle,AttributeType=S --key-schema
    AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE --
    ```

    `provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5`

   d. List tables again
   e. Describe the table to view the newly created table
      ```
      aws dynamodb describe-table --table-name Music --endpoint-url 
      http://localhost:8080
      ```
   f. Exit the container and restart it again
   g. List tables to see empty list again
   h. Mount a volume to the docker container to retain data
      i. Close and rerun container by adding a mount volume
         ```
         docker run -itp 8080:8080 -v dynamodb-local:/home/dynamodblocal/db db
         ```
   i. Retry creating a table and restarting container. Notice how data is persisted this time

6. Exercise 2 - Completing backend service code to connect to DB

1. Complete the code to call dyanmodb local in the file `exercises/backend-service/src/db_manager.py`
2. Complete put and list functions to call relevant db_manager functions inside the file `exercises/backend-service/src/server.py`
3. Add `boto3` to the list of requirements
4. Build and run a container
   a. Notice run might fail if db container is not up
5. Complete the Dockerfile provided under `exercises/db` if not already done as part of Demo 3 steps.
   ```
   FROM amazon/dynamodb-local
   WORKDIR /home/dynamodblocal
   RUN mkdir ./db && chown -R 1000 ./db
   EXPOSE 8080
   CMD ["-jar", "DynamoDBLocal.jar", "-dbPath", "./db", "-sharedDb", "-port", "8080"]
   VOLUME ["./db"]
   ```
6. Build and run a db container
   ```
   docker build . --tag db:latest
   docker run --name db -it -p 8080:8080 db
   ```
7. Restart backend service container again
8. Alternatively, instead of port forwarding we can link two containers to create a networking between them
   a. `docker run -it --name backend-service --link db:db backend-service`

7. Exercise 3 - Creating a webserver

1. Edit the configuration of nginx in `exercises/webserver/nginx.conf`
2. Complete the Dockerfile to pull base nginx image and copy necessary files
   ```
   FROM nginx:1.13-alpine
   COPY nginx.conf /etc/nginx/nginx.conf
   COPY index.html /usr/share/nginx/html/index.html
   ```
3. Review index.html file provided and make any necessary customizations
4. Build and run a container
   ```
   docker build . --tag webserver:latest
   docker run -it -p 80:80 --link backend-service:backend-service webserver
   ```
5. Go to a browser and navigate to `http://localhost`
6. Voila! We have an application setup and running

8. Exercise 4 - Creating a Docker compose for the container

1. Modify the existing docker-compose.yaml file provided at the root of the exercises directory to add the three containers under services
2. Add a volume for db
3. Configure networking to be host
    a. Experiment with bridge network
4. Build and run the container
   `docker-compose build`
   `docker-compose up`
5. To clean up resources after stopping the container run
   `docker-compose down`

9. Exercise 5 - Passing an Environment Variable to containers

1. Modify server.py of backend-service to read the db endpoint from an environment variable instead of hard coding it
2. Add the variable to docker-compose
3. Rerun containers

10. Demo 4 - Running container on cloud

11. Useful Docker commands

- Build an image from Dockerfile
  `docker build <PATH>`
- Run a command in a new container or start a new container
  `docker run <IMAGE>`
- Log in to a Docker registry
  `docker login`
- Log out from a Docker registry
  `docker logout`
- Attach local standard input, output, and error streams to a running container
  `docker attach <CONTAINER>`
- Save one or more images to a tar archive (streamed to STDOUT by default)
  `docker save <IMAGE>`
- Import the contents from a tarball to create a filesystem image
  `docker import <file|URL>`
- Stop one or more running containers
  `docker stop CONTAINER`
- Pull an image or a repository from a registry
  `docker pull <NAME[:TAG>`
- Push an image or a repository to a registry
  `docker push NAME[:TAG]`
- List containers
  `docker ps`
- List images

`docker images`

- Remove unused images
  `docker image prune`
- Remove one or more containers
  `docker rm <CONTAINER>`
- Remove one or more images
  `docker rmi <IMAGE>`
- Create a volume
  `docker volume create`
- Display detailed information on one or more volumes
  `docker volume inspect <VOLUME>`
- Remove all unused local volumes
  `docker volume prune`
- Return low-level information on Docker objects
  `docker inspect NAME|ID`
- Display a live stream of container(s) resource usage statistics
  `docker stats`
- Kill one or more running containers
  `docker kill <CONTAINER>`
- Create a network
  `docker network create <NETWORK>`
- Build, (re)create, start, and attach to containers for a service.
  `docker-compose up`
- Rebuild all images for a service
  `docker-compose build`
- Stop containers and remove containers, networks, volumes, and images created by up
  `docker-compose down`

12. Try on your own exercises

- Docker Orchestration Hands-On - https://training.play-with-docker.com/orchestration-hol/
- Docker includes swarm mode for natively managing a cluster of Docker Engines called a swarm. You can use the Docker CLI to create a swarm, deploy application services to a swarm, and manage swarm behavior. Interested in trying our Docker swarm mode?
  - Build your own Voting App
    - Tutorial: https://github.com/docker/labs/blob/master/swarm-mode/beginner-tutorial/README.md
    - Solution: https://github.com/dockersamples/example-voting-app
- Containerizing Machine Learning workflow
  - Tutorial: https://www.analyticsvidhya.com/blog/2021/06/a-hands-on-guide-to-containerized-your-machine-learning-workflow-with-docker/
  - Solution: https://github.com/dmoonat/sentiment-analysis/tree/master/containerized_webapp
- Interesting docker applications to experiment with -
  - FoodTrucks - https://github.com/prakhar1989/FoodTrucks
  - WordPress - https://docs.docker.com/samples/wordpress/

13. Resources

- Learn more about Boto3 - https://github.com/boto/boto3
- DynamoDB Local - https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.DownloadingAndRunning.html
- Nginx - https://www.nginx.com/resources/wiki/
- Python Flask - https://flask.palletsprojects.com/en/2.0.x/quickstart/