Containerizing Python App with Docker

- Sushan Kattel

What is Docker?

- Docker is analogous to a shipping container for software.
- Like shipping containers standardize global transportation of goods, Docker standardizes the packaging, shipping, and deployment of software applications.
- Key benefits of Docker:
  - Consistency across different environments
  - Scalability
  - Isolation
  - Portability

Why Containerize Python Apps?

- Improved Reproducibility
- Environment Isolation
- Deployment Consistency
- Dependency Management
- Compatibility Across Environments
- Simplified Maintenance and Deployment

Dockerfile Basics

- Dockerfile: A text document containing instructions to build a Docker image.

```
FROM python
WORKDIR /scrapper
COPY . /scrapper
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python3", "scrapper.py"]
```

➔ FROM: Specifies the base image for the container.

➔ WORKDIR: Sets the working directory inside the container.

➔ COPY: Copies files from the host to the container.

➔ RUN: Executes commands inside the container.

➔ CMD: Specifies the default command to run when the container starts.

docker-compose.yml file

- A file for defining multi-container Docker applications.
- Key components:
    - ***Services***: Each service represents a distinct component of the application, such as a web server or a database.
    - ***Build***: Defines how to construct the Docker image for a service, typically using a Dockerfile.
    - ***Image***: Specifies an existing Docker image to use for a service, which may be pulled from a registry like Docker Hub.
    - ***Ports***: Maps ports on the host machine to ports inside the container, enabling communication with services running inside Docker containers.
    - ***Volumes***: Facilitates sharing data between the host machine and containers or between multiple containers, providing persistent storage for applications.

Project 1

- Our first project is about writing csv file from python running inside docker container.
- Files:
  - Dockerfile
  - docker-compose.yml
  - requirements.txt
  - app.py
- Run it with: ***docker compose up*** command in terminal
- After completion, remove the container with command: ***docker compose down***

Project 2

- Our second project is about creating flask web app in docker container.
- Files:
  - Dockerfile
  - docker-compose.yml
  - requirements.txt
  - app.py
- Run it with: ***docker compose up*** command in terminal
- After completion, remove the container with command: ***docker compose down***

Project 3

- Our third project is about running multiple python files in docker container resembling our general projects scenario.
- Run it with: ***docker compose up*** command in terminal
- After completion, remove the container with command: ***docker compose down***

Saving the image file locally

- After the image is build with ***docker compose up***, we can save it locally as tar file.
- To save the image, run ***docker save -o <name to save in all lowercase>.tar <imagename>***
- You can then send the tar file to the user who wants to run it

Using the local image file

- After saving the provided tar file in a directory, load it to docker as: ***docker load -i <imagefilename>.tar***
    - *The -i flag in docker load -i img specifies reading image data from a file, useful for explicit file specification.*
- Now, run it with ***docker run --name <container name> -v "<local directory>:<docker directory>" <imagename>***
    - *You can escape **--name <container name>** if you don't want to specify the container name.*

Docker hub

- Docker Hub is a cloud-based repository provided by Docker, Inc., where you can store and manage Docker images.
- It serves as a central hub for Docker users to share and discover containerized applications, making it easier to collaborate and distribute Docker images.

Pushing Image to Docker Hub

- Build the image with ***docker build -t <nameofimage>:<versionofimage> .***
  - The version of image is default to ***latest*** if not defined.
  - Don't miss the **.** at end. It specifies the build context.
- To push a Docker image to Docker Hub, we first need to tag the image with the repository name and version.
  - ***docker tag <nameofimage>:<versionofimage> <username>/<nameofimage>:<versionofimage>***
- Now push the image to repository with ***docker push***
  - Remember, you need to be logged in before pushing the image. Use ***docker login*** or follow this guide for login (if ubuntu)

Using Image from Docker Hub

- Using Docker images from Docker Hub is straightforward. We simply pull the desired image using the ***docker pull*** command as ***docker pull skattel/fuse_testpython:v1.0.0.*** (<username>/<imagename>:<version>)

- As the container is an isolated environment, the data are not persistent and are only inside the container.
    - If there is no data to save as output from the script, we can directly run the images as: ***docker run <imagename>:<version>***
    - If the data is to be saved in a persistent storage, we need to mount the storage and use docker run as: ***docker run -v "<localdirectory>:<containerdirectory>" <imagename>:<version>***

        Eg: ***docker run -v "$(pwd)/output:/app/output" skattel/fuse_testpython:v1.0.0***

Purging Containers and Images

- Cleaning up unused containers and images is essential for maintaining system performance, freeing up disk space, and ensuring a tidy Docker environment.
- We can use the ***docker container prune*** command to remove stopped containers and the ***docker image prune*** command to delete unused images.
- To delete the specific containers, proceed as follows:
    - ***docker ps -a*** : Lists all the containers (running or stopped).
    - ***docker rm <containername>*** : Removes the specified container.
    - ***docker images*** : Lists all the images stored locally on the system.
    - ***docker rmi <imagename>:<ver>*** : Removes the specified image.

Thank you !