Learning Docker

Table of contents

bit.ly/Learning-Docker

What is Docker?

“Docker is an open platform for developing, shipping, and running applications.”

“Docker provides the ability to package and run an application in a loosely isolated environment called a **container.** ”

bit.ly/Learning-Docker

Why use Docker and Containers?

Self-contained: Has everything needed to run the application without relying on what is currently installed in the host:

- Allows us to have standardized environments.
- Easier to share and set up in new machines.
- Provides isolation and security.

Installing Docker

The github repo has three bash scripts:

- 0-Utils/install-docker
- 0-Utils/install-docker-compose
- 0-Utils/docker-post-installation

In the terminal, do:

```
chmod +x <script>
```

```
<script>
```

In VS Code: Add the Docker Extension

Docker Image

“An image is a **read-only template** with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.”

Ex: an image based on ubuntu with conda installed.

Docker Image

Has instruction and files + metadata to create a system

- Files form the root filesystem
- Commands to execute when starting the container
- Environment variables

Docker Image

Multiple containers can be created based on an image.

An image is conceptually similar to a *Class*

A container would be similar to an *instance*

Docker Image

If an image is read-only, how do we change it?

We don’t, we create a new image from the image of interest with all the changes we want.

How?

Dockerfile

Instructions to create an image. Each instruction in a Dockerfile creates a layer in the image.

Building images is a **fast and lightweight** virtualization process:

- When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt

Dockerfile - Instructions

- ARG defines a variable that users can pass at build-time.
- FROM indicates the base image for our build.
- WORKDIR sets the working directory.
- RUN lines will be executed by Docker during the build.
  - Our RUN commands must be non-interactive

Dockerfile - Instructions

- EXPOSE informs Docker that the container listens on the specified network ports at runtime.
- ENV sets the environment variable <key> to the value <value>. These will persist when a container is run from the resulting image.

  -
    ```
    ENV MY_NAME="Anna"
    ```

Dockerfile - Instructions

- COPY copies new files or directories from <src> and adds them to the filesystem of the image at the path <dest>.
  - `COPY home /mydir/`
  - *src*: paths are interpreted as relative to the source of the context of the build.
  - *dest*: absolute path, or a path relative to `WORKDIR`
- ➢ If you need multiple files from your context, COPY them individually, rather than all at once. This ensures that each step's build cache is only invalidated (forcing the step to be re-run) if the specifically required files change.

Dockerfile – Instructions

- ENTRYPOINT sets the image's main command that will be executed everytime we run the command.

RUN vs ENTRYPOINT:

RUN is only executed once: When we are building the image and we can have multiple RUN commands.

ENTRYPOINT is executed everytime we run the container and it can appear only once in the Dockerfile.

Dockerfile - Instructions

- VOLUME creates a mount point with the specified name and marks it as holding externally mounted volumes from native host.
  - We are going to learn more about volumes later.

Let's take a look at the example. (folder 1-Dockerfile)

Docker Compose

- Useful when:
  - We have multiple containers we want to connect or run together (e.g: Train, Re-train, Test, Predict). We will call them *services*.
  - We require particular run parameters.
- Not useful to:
  - Be used as an orchestrator.
  - Be used with Kubernetes.

Docker Compose - Services

Docker-compose files specify how each service will be configured. Some things we can configure are:

- build specifies the build configuration for creating the container image from source.
  - Dockerfile
  - Context
- image specifies the image to start the container from.

Docker Compose - Services

- container_name specifies a custom container name
- volumes defines mount host paths or named volumes that will be accessible by service containers. (more on mounts and volumes later)
- command overrides the the default command declared by the container image

Docker Compose – Services

- deploy specifies configuration for the deployment and lifecycle of services and runtime requirements. Some elements we can configure include:
  - resources: physical resources e.g. gpu
  - cpus: limit or reservation of cpu cores
  - replicas: number of containers that should be running at any given time.

Docker Volumes and Bind Mounts

Writable data for a container is ephemeral and it will be lost when the container is removed.

We can add persistent data to a container using *Docker volumes* or *Docker bind mounts*. They will store files on the local host filesystem.

Docker Volumes

- Managed by Docker
- Located in Docker's "private storage area.
  (var/lib/docker/volumes/<volume ID>/)
- Don't need to check directory exists before mounting it. Better for portability.
- No problems with permissions. Docker owns the volume.

Docker Bind Mounts

- Managed by user/developer
- Directory can be located anywhere in the local system.
- Directory needs to exist on the local system. User will need to create it if it doesn't.
- Might encounter permission issues.

Can be mounted to multiple containers

Docker Compose

Let's take a look at the example. (folder 2-Docker-compose)

Exposing the GPU

We need to configure our local system by installing the nvidia-container-runtime

The github repo has a bash script for this:

- 0-Utils/GPU-enable-for-Docker

In the terminal, do:

```
chmod +x 0-Utils/GPU-enable-for-Docker
```

```
0-Utils/GPU-enable-for-Docker
```

Exposing the GPU

Add device to Docker-compose

```
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: 1
            capabilities: [gpu]
```

Change Docker file to use image with GPU drivers/support

Exposing the GPU

Let's take a look at the example. (folder 3-Docker-GPU)

Questions?

Thank you