Docker Essentials

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.**

- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

What can I use Docker for?

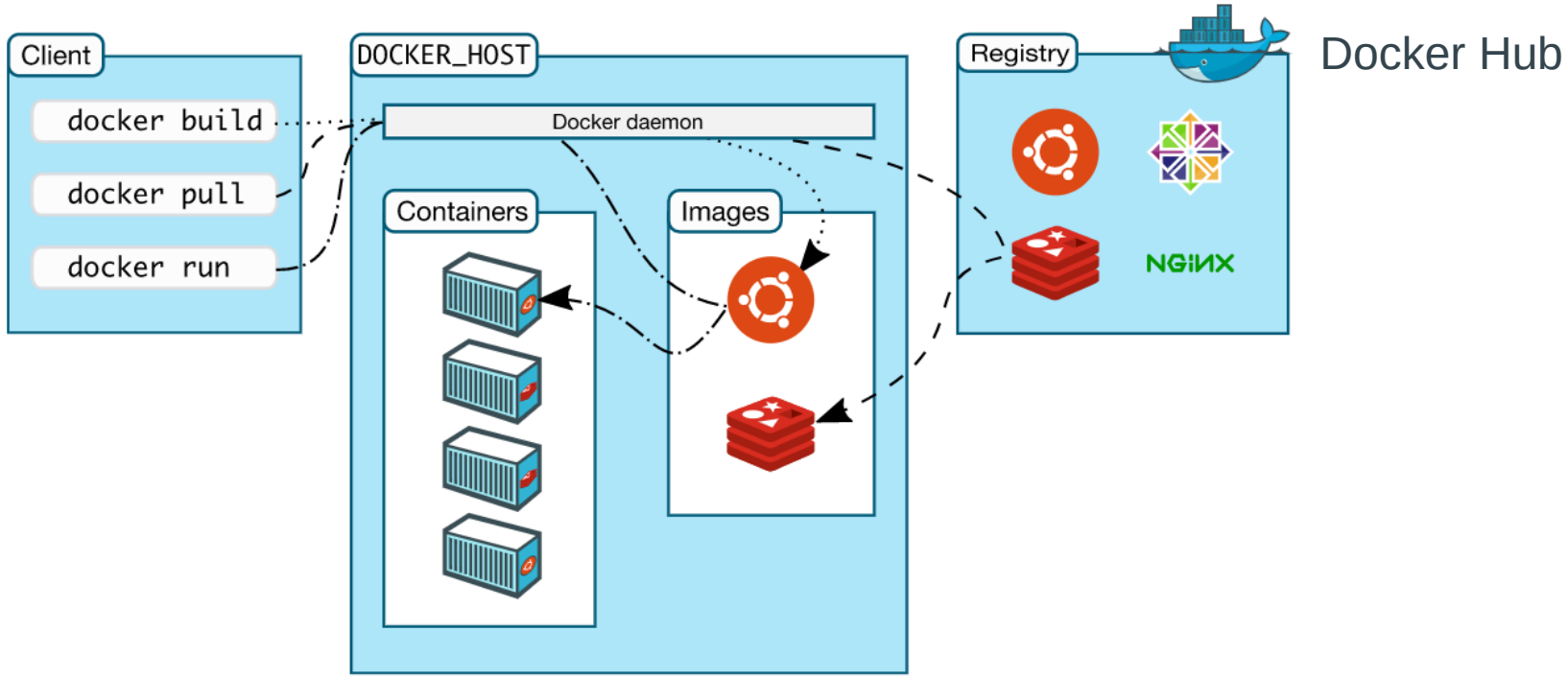
- Docker **streamlines the development lifecycle** by allowing developers to work in standardized environments using local containers which provide your applications and services.

- Containers are **great for continuous integration** and continuous delivery (CI/CD) workflows.

- Docker's portability and lightweight nature also make it easy to dynamically manage workloads, **scaling up or tearing down** applications and services as business needs dictate, in near real time.

Docker architecture

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.



The Docker daemon: dockerd

- Listens for Docker API requests and **manages Docker objects such as images, containers, networks, and volumes**.
- A daemon can also communicate with other daemons to manage Docker services.

The Docker client: docker

- Is the primary way that many Docker users interact with Docker.
- When you use commands such as docker run, the client **sends these commands to dockerd**, which carries them out.
- The docker command uses the Docker API.
- The Docker client can communicate with more than one daemon.

- **The docker client can be a desktop application or a cli**.

Docker registries

- A Docker *registry* stores Docker images.
- **Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default**.
- You may run your own private registry.

- When you use the docker pull or docker run commands, the required images are pulled from your configured registry.
- When you use the docker push command, your image is pushed to your configured registry.

Docker image and Dockerfile

- An *image* is a read-only template (file) with **instructions for creating a Docker container**.
- Often, an image is *based on* another image, with some additional customization.
- You might create your own images or you might only use those created by others and published in a registry.
- To build your own image, you create a ***Dockerfile*** with a simple syntax for defining the steps needed to create the image and run it.
- Each instruction in a Dockerfile creates a layer in the image.

Docker container

- A container is **a runnable instance of an image**.
- You can create, start, stop, move, or delete a container using the Docker API or CLI.
- You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine.
- **A container is defined by its image as well as any configuration options you provide to it when you create or start it.**

Docker Compose

- Is a tool you can use to **define and share multi-container applications**.
- With Compose, you can create **a YAML file,** `docker-compose.yml`, to define the services and with a single command, can spin everything up or tear it all down.
- Is installed with Docker Desktop/Toolbox for either Windows or Mac. But if you are on a Linux machine, you will need to install Docker Compose.

- To test if you have Docker compose:

```
docker-compose version
```

Difference between Docker Compose Vs Dockerfile

- A Dockerfile is a simple text file that contains the commands a user could call to assemble an image whereas Docker Compose is a tool for defining and running multi-container Docker applications.

- Docker Compose define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.

- Your Docker workflow should be to build a suitable Dockerfile for each image you wish to create, then use compose to assemble the images using the build command.

The underlying technology

- Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality.
- Docker uses a technology called namespaces to provide the isolated workspace called the container.
- When you run a container, Docker creates a set of namespaces for that container.
  - These namespaces provide a layer of isolation.
  - Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

References & Links

- https://docs.docker.com/get-started/overview/
- Getting started guide
  https://docs.docker.com/get-started/

- Use Docker Compose
  https://docs.docker.com/get-started/08_using_compose/