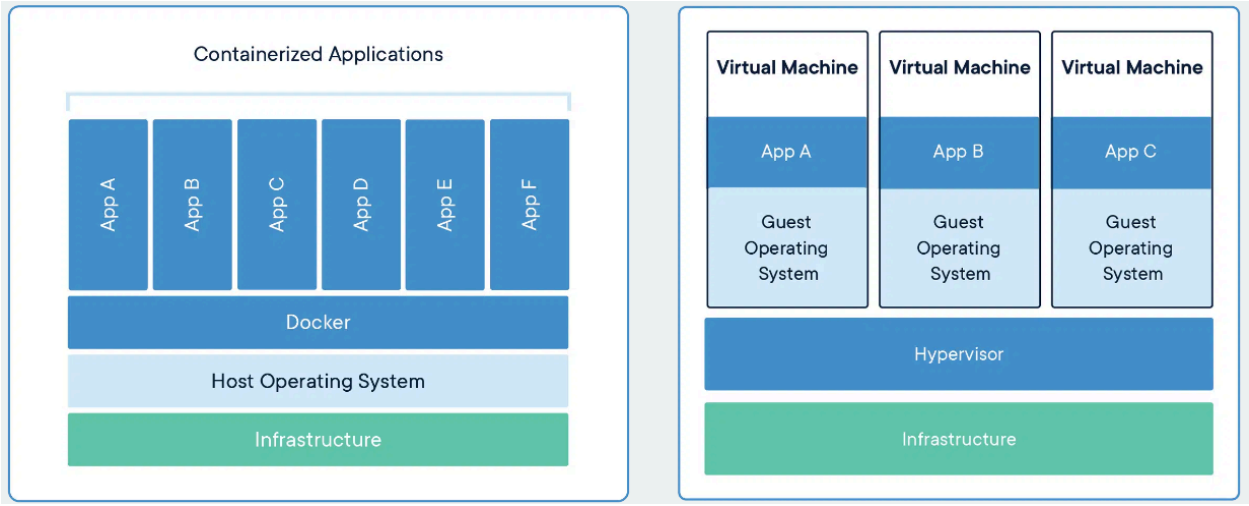
Containerization & Docker

Containerization

In recent years, a technology known as "containerization" has become very popular. As a concept, it refers to packaging up all that is required to run an application, including runtime dependencies, configuration files, and settings, and then running the application in an isolated, pre prepared environment. This is similar to running separate applications in entirely separate virtual machines, but is generally much more lightweight, as the containerized application is run directly on the host operating system, and is simply isolated using features the host already provides.



Docker

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Containers can be created and built on-site when used to run your own software, or you can make use of already published containers for third party dependencies, hosted on public container registries, the most well known of which is likely Docker's own hub.

We must be familiar with some terminologies to get started with Docker.

- Container - For encapsulating the required software, we need this running instance.
- Image - images are pre-built packages containing an operating system, software, and configurations - like a blueprint for your app.
- Port - A TCP/UDP port. This can be exposed to the outer world (accessible from the host OS) or connected to the other containers.
- Volume - The shared folder can be described through this.
- Registry - It is the docker image storing server

Workflow of Using Docker::

- When starting using Docker, we would have an application ready to be deployed, the data, and the trained model.
- We would then write the code for something known as Dockerfile, which is nothing but a plain text file that contains instructions that docker uses to pack your app into an image.
- Once a dockerfile is ready, the next step would be to build an image.
- For using this image, we would run the image that will basically create a container that will run your app.

Dockerfile

To build a Docker image, you need to create a Dockerfile. It is a plain text file with instructions and arguments. Here is the description of the instructions we're going to use in our next example:

- FROM — set base image, docker offers a lot of pre-built containers for the developers to use, so that they don't have to start again from scratch.
- RUN — execute command in container like upgrade pip and pip install
- ENV — set environment variable
- WORKDIR — set working directory
- VOLUME — create mount-point for a volume
- CMD — set executable for container

You can check the Dockerfile reference for more details.
Example:

```
FROM python:3.8-slim-buster

WORKDIR /flask-docker
```

```
RUN python3 -m pip install --upgrade pip
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```