CS2043: Docker

Friday 22, 2019

Virtualization, Containers and VMs

- Virtualization is the creation and use of a virtual (logical) version of a resource
- People often use Virtual Machines (VMs) to use multiple operating systems on a single machine without any additional hardware
  - For example, run a copy of MacOS on your Windows 10 machine
  - Relies on a piece of software called a “Hypervisor” to create and manage virtual versions of computer hardware (processor, memory, network card, etc.)
  - Create the illusion that the VM operating system has exclusive access to physical hardware
- Containers are a much lighter form of virtualization - they virtualize the environment in which a process runs (runtime environment)
  - Package all code and dependencies so that you can run your program on any system
  - Provides isolation from other processes running on the same machine

Problem Development to Production Delay

Imagine yourself as a web developer who is developing a new feature. During the development phrase, you created a new feature in a development environment (let’s say your laptop).

Later when you want to integrate that feature into your web application, you would have to apply the changes to your production environment (let’s say your server in the cloud). This can be a nightmare! You must make sure that the code will also run flawlessly in the new environment. What if package X that your feature relies on is version 2.3 locally but is version 2.4 on the clouds and your feature no longer works? **Let Docker come to the rescue.**

What is Docker Pictorially?

What is Docker? Overview

- “Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.”
- Concept: build once run anywhere.
- Pro: reduce the delay between development and production.

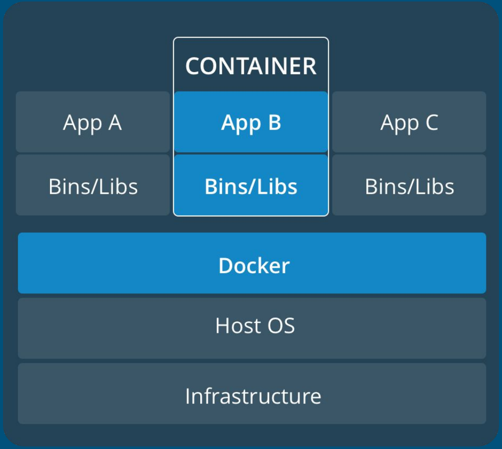
Why is Docker Useful?

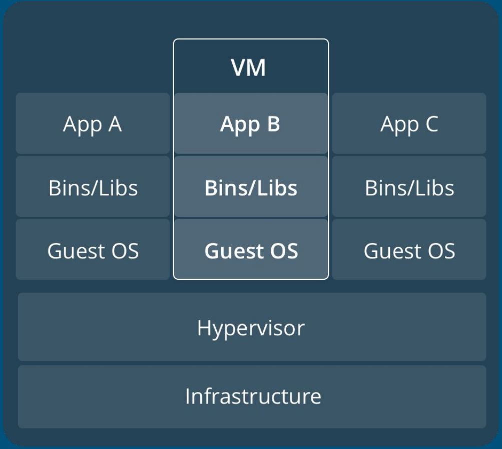
- **Standardized environment** leads to faster development.
- Great for continuous integration and continuous delivery (CI/CD) workflows.
- **Highly portable** workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.
- **Lightweight** and Fast Compared to VMs.

Docker vs VM

Differences

- No hypervisor (Virtual Machine Monitor).
- Run directly on Host Machine's Kernel.
- Each container run is a discrete process, takes up about the same amount of memory as any other executable.
- Light-weight

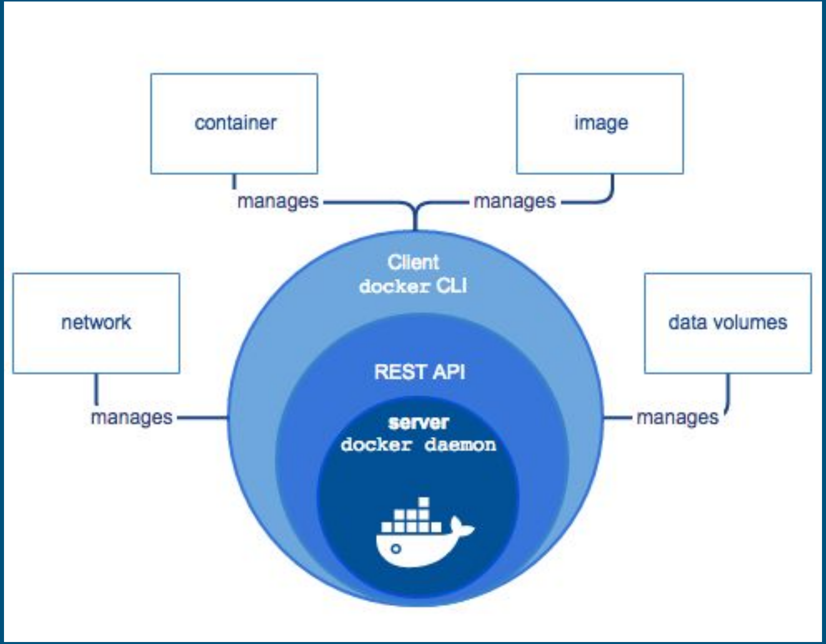




Docker Engine

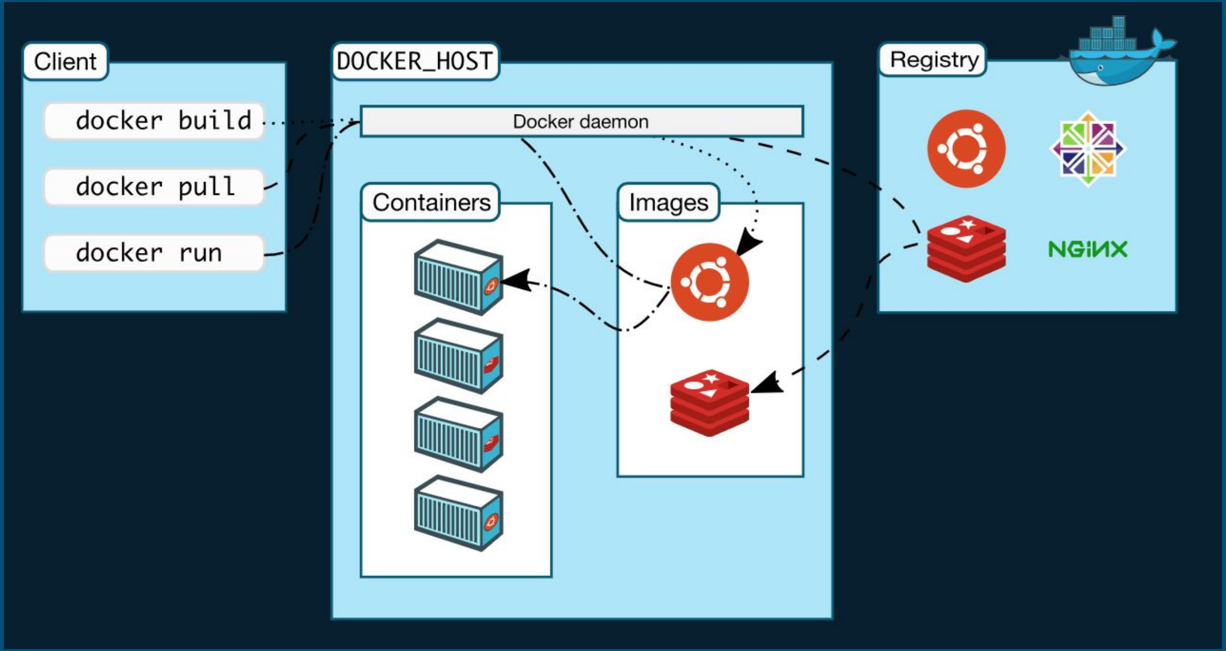
Client-Server Architecture with three components.

1. **Command Line Interface** (CLI): the \`docker\` command.
2. **REST API**: talks to the server daemon.
3. **Server Daemon**: long-running background process.



Docker Architecture

This is the big picture. More details are included in the following slides.



Docker Images

- An image is an executable package that includes everything needed to run a container: the code, a runtime, libraries, environment variables, and configuration files.
- 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. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Docker General and Image Commands

`docker info` = General Information

`docker image ls` = List Images

`docker image rm <IMAGE ID>` = Removes Image

`docker build -t <Image Name> .` = Creates an image named <Image Name> from the Dockerfile in the current directory.

Dockerfile Example

```dockerfile
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Dockerfile Commands

`FROM <image name>` = Sets the base image for subsequent instructions.

`RUN <command>` = The RUN instruction will execute any shell commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile. This command is mainly used for installing packages. This command runs once when building the image but does not run when the image runs.

`CMD ["executable","param1","param2"]` = Sets the command to be executed when running the image. ONLY ONE per Dockerfile.

Dockerfile Commands 2

`WORKDIR /path/to/workdir` = The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.

`COPY <src> <dest>` = The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

`EXPOSE <port number>` = The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

Docker Containers

- 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. (Container networking and storage will not be covered in this lecture)

Docker Container Commands

`docker run -p <local port number>:<container port number> <image name>`

Creates a container from the image <image name> and forward traffic from port <local port number> to port <container port number>. If the image does not exist on your machine. The Docker CLI will pull it from Docker Hub. (port definition: an endpoint of communication that is identified by a number.)

`docker run -d -p <local port number>:<container port number> <image name>`

Same as above but runs the container in the background. The `d` flag stands for detached mode.

Docker Container Commands 2

`docker container ls` = List all running containers

`docker container ls -a` = List all containers

`docker container stop <name or id>` = stops the container with the specified container name or id.

`docker container rm <name or id>` = removes the container with the specified container name or id.

Docker Registry

- A docker registry stores images.
- Docker Hub is the most popular public registry (Github for Dockers).
- Docker images can be pulled from or pushed to Docker Hub in a similar fashion to pushing or pulling from Github.

Docker Registry Commands

`docker login` = CLI login to your Docker Hub. Must log in before pushing.

`docker tag <image> username/repository` = Tag <image> for upload to registry

`docker push username/repository` = Upload tagged image to registry

`docker run username/repository` = Run image from a registry

Container Activity

`docker run -i -t ubuntu`

- Creates an interactive ubuntu container.
- Execute `exit` to exit from the container.

`docker run -i -t python`

- Creates an interactive python container.
- Execute `exit()` to exit from the container.

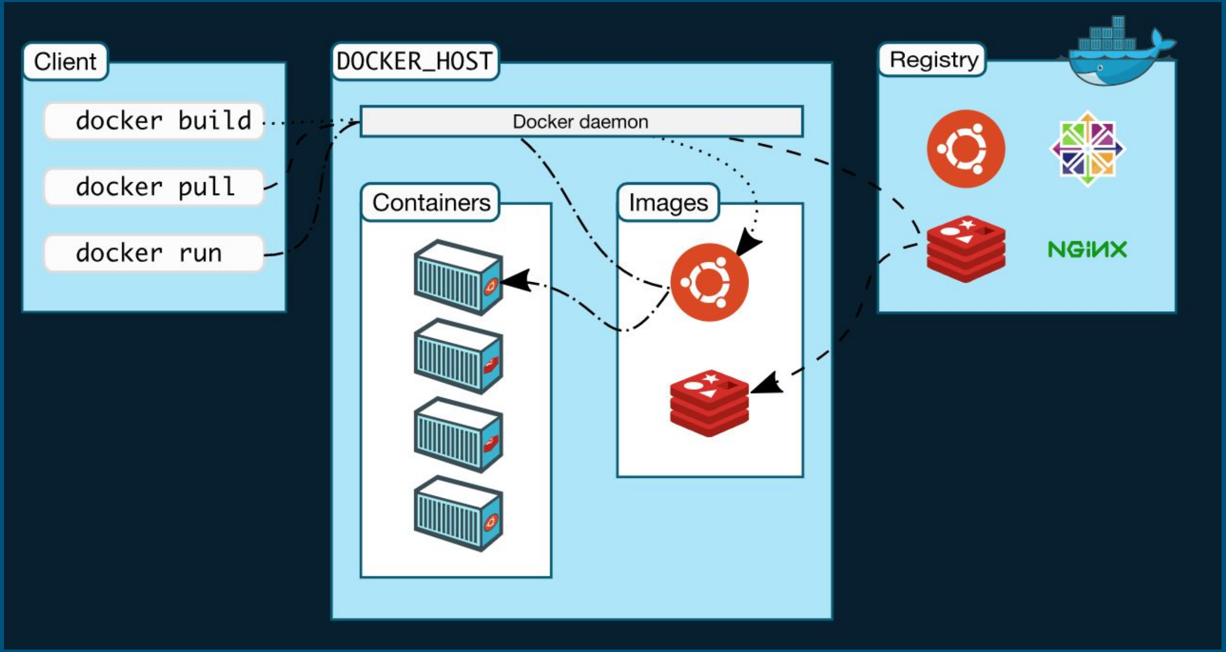
Flask Deployment Demo

If you did not come to class, you can play with this demo by yourself over break.

Link: https://docs.docker.com/get-started/part2/

Docker Architecture Revisited

Hopefully this diagram makes more sense :D



More Information

For More Information on how Docker Works

https://docs.docker.com/engine/docker-overview/

For More Information on Dockerfiles

https://docs.docker.com/engine/reference/builder/

Google Cloud Platform (Compute Engine)

Google Compute Engine delivers virtual machines running in Google's innovative data centers and worldwide fiber network.