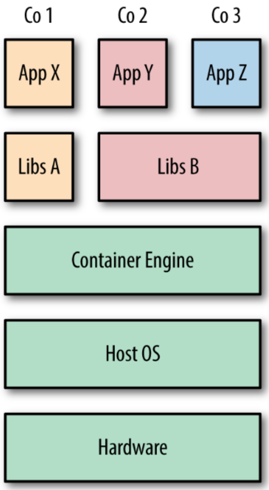
CS200
Software Tools & Technologies Lab II

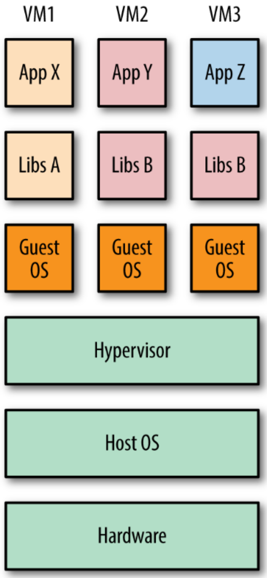
Session 9

Building Docker Images

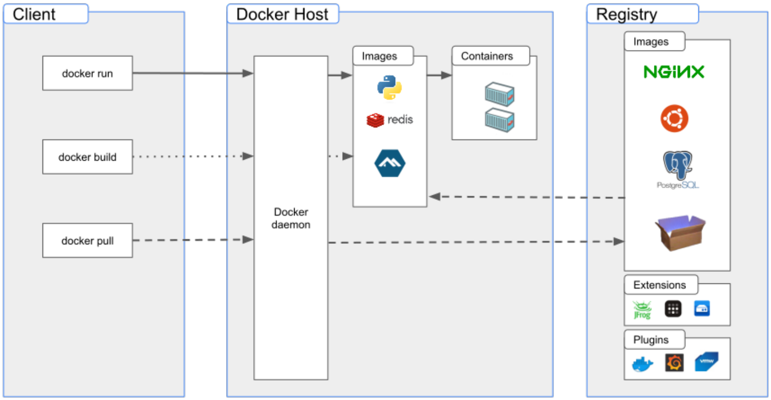
Instructor
Dr. Dhiman Saha

Containers Vs Virtual Machines





Docker Architecture



Registries, Repositories, Images, and Tags

Registry

A service responsible for hosting and distributing images. The default registry is the **Docker Hub**

Repository

A collection of related images (usually providing different versions of the same application or service)

Tag

An alphanumeric identifier attached to images within a repository (e.g., 14.04 or stable).

`docker pull iit/bhilai:latest`

$$\overbrace{\texttt{docker}}^{master-command}\quad\overbrace{\texttt{pull}}^{sub-command}\quad\overbrace{\texttt{iit/bhilai}}^{repository}:\overbrace{\texttt{latest}}^{image-tag}$$

- Will download the image tagged `latest` within the `iit/bhilai` repository from the **Docker Hub** registry.

EXP-8 Keep The Change (docker commit)

- Recall EXP-2

```
docker run -i -t debian /bin/bash

root@645e033e0088:/# apt-get update && apt-get
install iputils-ping
```

- How to make the change permanent and re-use this type of container as a basis for others?
- On the docker host run the following

$$\texttt{docker commit } \overbrace{\texttt{645e033e0088}}^{container-ID} \ \overbrace{\texttt{debian:ping}}^{tag}$$

```
sha256:7a11db7ba1c0713bde5e74856dabf4ac3f07593f93fe3cd0c6e3f53fc709dbc7
```

EXP-9 Save Thy Image/Container

Problem

You have created some images or have some containers that you would like to keep and share with your collaborators.

- `docker save` and `load` to create a tar ball from a previously created image
- `docker import` and `export` for containers

export squashes history — **Containers**

```
docker export 77d9619a7a71 > update.tar
docker import - update < update.tar
```

save preserved history — **Images**

```
docker save -o update1.tar update
docker load < update1.tar
```

EXP-10 debian:ping with Dockerfile

Your First Dockerfile

- `mkdir ping`
- `cd ping`
- `touch Dockerfile`

Inside Dockerfile

```
FROM debian:latest
RUN apt-get update && apt-get install -y iputils-ping
```

- Build the image
  `docker build .`

- Fire it up!!!
  `docker run <Image-ID> ping google.com`

EXP-11 Revisiting Hello-World

- Write a Dockerfile with following content.
  ```
  FROM debian
  ENTRYPOINT [‘‘/bin/echo’’, ‘‘Hello World’’]
  ```
- Now build it.
  ```
  docker build -t test/hello .
  ```
- Then run it.
  ```
  docker run test/hello
  ```

Ger More

This container is pretty useless. Why?

- The `ENTRYPOINT` value cannot be overwritten this image will only be able to run echo
- How to make this more useful?

Useful Hello-World

- Write a Dockerfile with following content.

```
FROM debian
CMD [‘‘/bin/echo’’, ‘‘Hello World’’]
```

- Now build it.

```
docker build -t test/hellonew .
```

- Then run it.

```
docker run test/hellonew
```

CMD

Defines the default behavior.

- Run another command

```
docker run test/hellonew /bin/date
```

Dockerfile

- A Dockerfile is a text file that represents the way a Docker image is built.
- It also captures what happens when a container is started with this image.
- Starting with three simple instructions you can build a fully functioning container:
  - `FROM`
  - `ENTRYPOINT`
  - `CMD`
- Remember that `CMD` can be overwritten by an argument while `ENTRYPOINT` cannot.
- A process we want to run in a container needs to run in the foreground, otherwise the container will stop.

Executing a script

- In-Class Assignment - Write a script “entrypoint.sh”

```
FROM debian

RUN <pre-requsites for your script>

COPY entrypoint.sh /

ENTRYPOINT ["/entrypoint.sh"]
```

COPY

- The COPY instruction simply copies a file from the host into the images filesystem.
- The first argument is the file on the host
- The second the destination path inside the image

A Flask App inside a container

- Use the simple Hello World application defined by the following Python script.

```
#!/usr/bin/env python

from flask import Flask

app = Flask(__name__)
@app.route('/hi')

def hello_world():
   return 'Hello World!'

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=5000)
```

Dockerfile

```
FROM debian
RUN apt-get update
RUN apt-get install -y python3
RUN apt-get install -y python3-pip
RUN apt-get clean all

RUN pip3 install flask

ADD hello.py /tmp/hello.py

EXPOSE 5000

CMD ["python3","/tmp/hello.py"]
```

EXP-14 Fire it Up!!!

```
docker build -t flask .
```

```
docker run -d -P flask
```

docker ps

```
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS         PORTS                                              NAMES
680d0759eee5   flask   "python3 /tmp/hello."    5 seconds ago    Up 4 seconds   0.0.0.0:49154− >5000/tcp, :::49154− >5000/tcp      condescending_goldwasser
```

- PORTS shows a mapping between port 5000 of the container and port 49154 of the Docker host[1].
- See it in action from Docker host!
  - Simple curl to `http://localhost:49154/hi`
  - Or open your browser to the same url.

[1]This might be a different port in your case.