Docker - First Step

Exercise

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

- Requirement: Docker

- Get it from https://hub.docker.com/editions/community/docker-ce-desktop-mac

Images and Containers

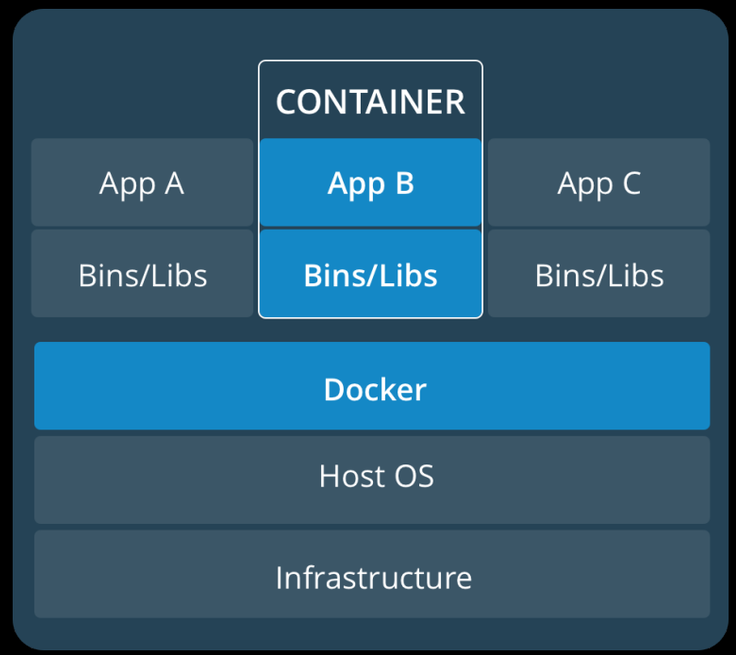
A container is launched by running an **image**. An **image** is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.

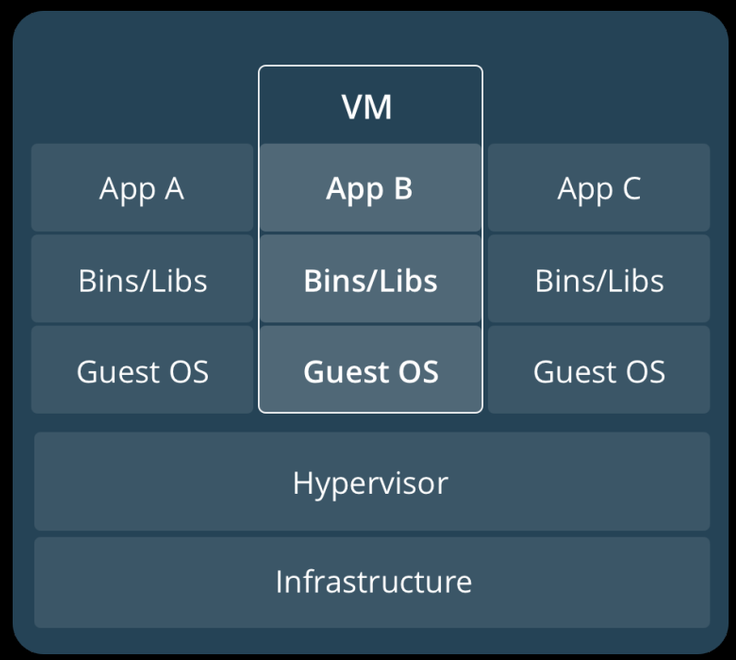
A **container** is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, `docker ps`, just as you would in Linux.

Containers and virtual machines

A **container** runs *natively* on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a **virtual machine** (VM) runs a full-blown “guest” operating system with *virtual* access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.





Check Docker Environment

1. Test docker version:

   > `docker --version`

   Docker version 18.09.2, build 6247962

2. More information :

   > `docker info`

   Containers: 7
   Running: 1
   Paused: 0
   Stopped: 6
   Images: 408
   Server Version: 18.09.2
   Storage Driver: overlay2
   ….

Test Docker installation

```
> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

What happened

- Docker run <image>
  - Starting a container from <image>
    1. Checking for <image> locally
    2. Checking for <image> at image repository, default hub.docker.com
- Executing “program” inside container

List Docker Images

- List all images created or downloaded to your machine

```
docker image ls
```

List container

- Running container

  `docker container ls`

- Running or stopped container

  `docker container ls –all`

- Alias

  `docker ps –a`

Cheat Sheet

- List Docker CLI commands

  docker
  docker container —help

- Display Docker version and info

  docker —version
  docker version
  docker info

- Execute Docker image

  docker run hello-world

- List Docker images

  docker image ls

- List Docker containers (running, all, all in quiet mode)

  docker container ls
  docker container ls --all
  docker container ls -aq

Container as development environment

- Previously we executed a container as a binary

  ```
  docker run hello-world
  ```

- Starting a container and work interactively

  ```
  docker run -t -i ubuntu:latest bash
  ```

- `docker run -t -i ubuntu:latest bash`

  `-t  : Allocate a pseudo-TTY`

  `-i  : Keep STDIN open even if not attached`

  `-rm : Automatically remove the container when it exits`

- Installing software

    1. apt-get -y update && apt-get -y upgrade
    2. apt-get -y install git
    3. git clone https://github.com/E3SM-Project/cime.git
    4. cd cime ; ./scripts/create_newcase
    5. ……..

Creating an Image

- Two options:
  - Install software in running container and make a snapshot
  - Create reproducible image from repeatable install instructions (Dockerfile)

apt-get install libxml-libxml-perl

Dockerfile

`Dockerfile` defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to “copy in” to that environment. However, after doing that, you can expect that the build of your app defined in this `Dockerfile` behaves exactly the same wherever it runs.

Creating a Dockerfile

1. Create an empty directory and change into it

2. Create a new file called **Dockerfile**

```
# Use an official alpine runtime as a parent image
FROM alpine:latest
# Set the working directory to demo
WORKDIR /demo
# Copy the current directory content into the
container ad /demo
COPY . /demo
# Install the needed packages
RUN apk add git \
    perl \
    perl-xml-libxml
RUN git clone https://github.com/E3SM-Project/cime.git
# Define environment variable
ENV NAME Demo
# Run echo when the container launches
without arguments
CMD [ "echo" , "Welcome"]
```

Building an Image

```
> docker build -t demo:latest .

Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM alpine:latest
 ---> 4d90542f0623
Step 2/7 : WORKDIR /demo
 ---> Using cache
 ---> ce6ba6e7552a
Step 3/7 : COPY . /demo
 ---> 182b7eeea1f3
Step 4/7 : RUN apk add git     perl     perl-xml-libxml
 ---> Running in cfa7d7a11c61
fetch http://dl-cdn.alpinelinux.org/alpine/v3.10/main/x86_64/
APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.10/community/
x86_64/APKINDEX.tar.gz
(1/16) Installing ca-certificates (20190108-r0)
(2/16) Installing nghttp2-libs (1.38.0-r0)
```

Examine

- docker run --rm demo:latest
- docker run --rm demo:latest cime/scripts/create_newcase
- docker run -ti --rm demo:latest ash
- docker run -ti --rm -v `pwd`:/from-host demo:latest ash