

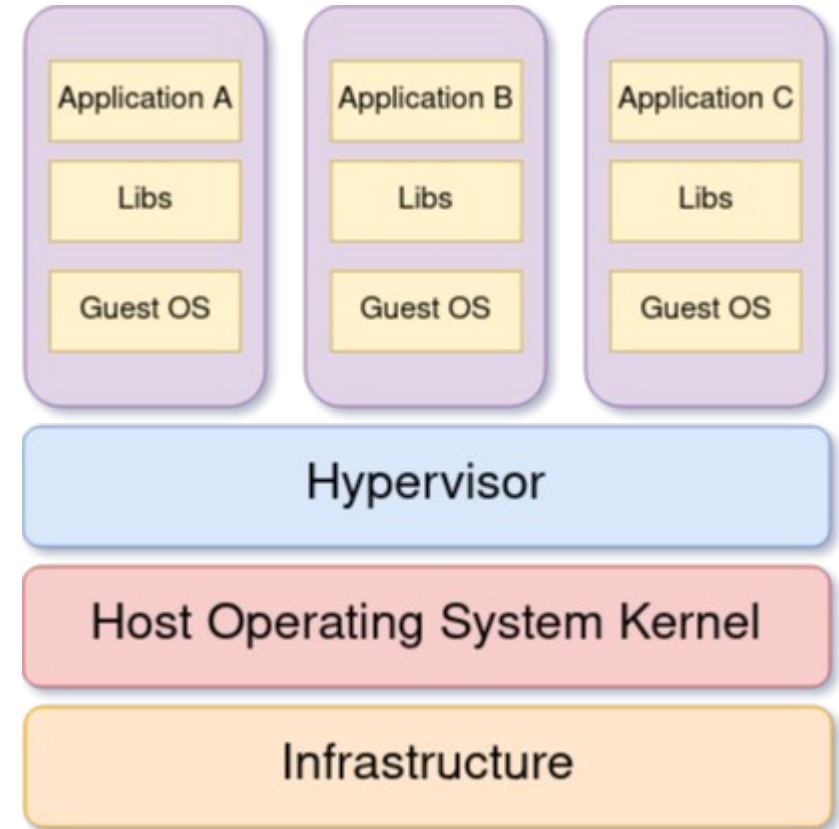
Full Stack Development

Containers, Microservices and UI

2. Docker Containers

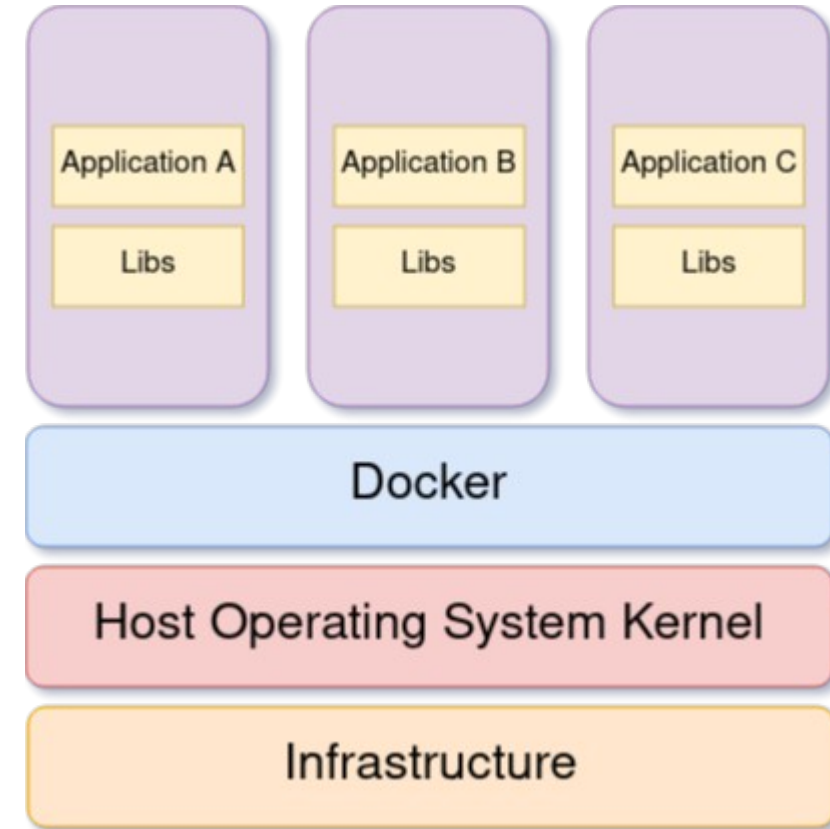
Virtual Machines

- Each VM is a full installation of a complete OS
 - The VM hard drive maps to a file or files on the host OS
 - The guest OS hardware calls are relayed to the host OS by the hypervisor
- VMs are slow to start and have a large footprint
 - Great for emulating a computer
 - Too heavyweight for running a small lightweight process



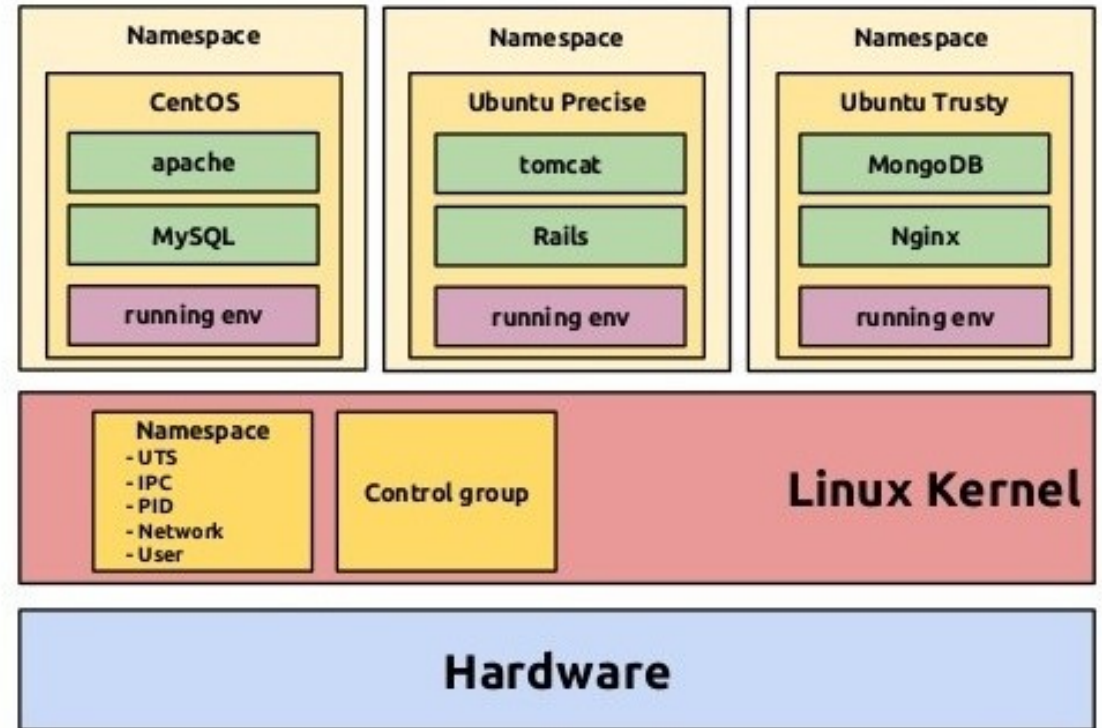
Containers

- A container is a lightweight process managed by the Docker engine
 - It has no persistent storage
 - Contains only what is needed to run the application
 - Small footprint
 - Fast start up and shutdown
- Based on Linux containers
 - Use specific features of Linux the Linux kernel
 - Windows can run containers by using an embedded Linux VM

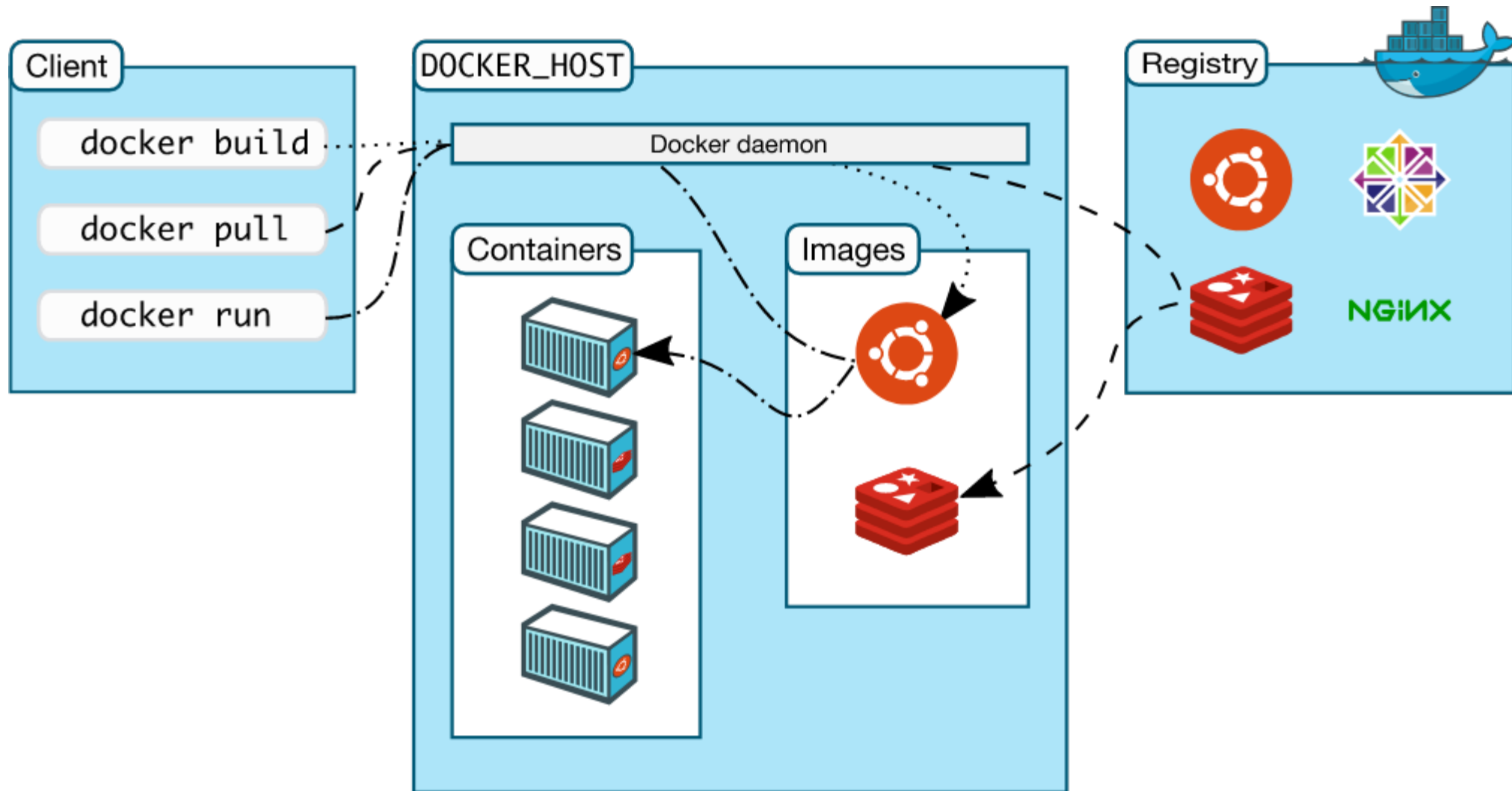


Implementation

- Linux containers run in isolated environments using Linux namespaces and control groups
- Provide resource limitation, prioritization, accounting, and control
- Hides the process space and resource information of each container from the others
- Docker is an implementation of Linux containers



Docker Architecture



Docker Terminology

- Docker Daemon or Engine
 - The process that manages images and containers on the Docker host
 - The Docker CLI is used to request services from the Docker engine
- Docker Image
 - Analogous to an executable file - template for running a container
- Docker Container
 - A Docker image that is executing or has finished executing
 - Analogous to a process that is running an executable file
 - Multiple containers can be created and run from a single image
- Docker Registry
 - A version collection Docker images
 - Each set of versions for an image is referred to as an image repository

Docker Images

- Docker images are read only
 - Uniquely identified by hash codes
- Built-up in layers
 - Uses Linux union file system, also referred to as an overlay file system
 - Each layer is immutable identified by a unique hash code
 - Layers are shared by images – only one copy of a layer exists



Docker container
(AUFS storage-driver demonstrating whiteout file)

Docker Registry

- The local registry
 - On the machine running Docker
 - Images pulled from other registries are cached here
 - This is the first registry searched for a requested image
- Docker Hub
 - Public repository maintained by Docker
 - Searched by default after the local registry
- Other registries
 - Docker can be configured to use other registries
 - Allows control over which images Docker pulls
 - Ensures only approved images are used by Docker installation

Docker Repository

- Images are versioned
 - A set of versioned images is called repository
 - A specific image is referenced by <image_name>:<version_tag>
 - The following are different versions of the Ubuntu image
 - ubuntu:18.04
 - ubuntu:20.04
 - If no version tag is specified, then the version defaults to “latest”
 - Pulling the image **ubuntu** is the same as pulling **ubuntu:latest**
- Images are uniquely identified by their digest value
 - Tags are identifies that are added for convenience
 - Images do not have to have tags but a single image can have multiple tags
 - Images can only be deleted if they have zero or one tags

Exploring Docker Registries

Demo



Working with Docker Images

Demo



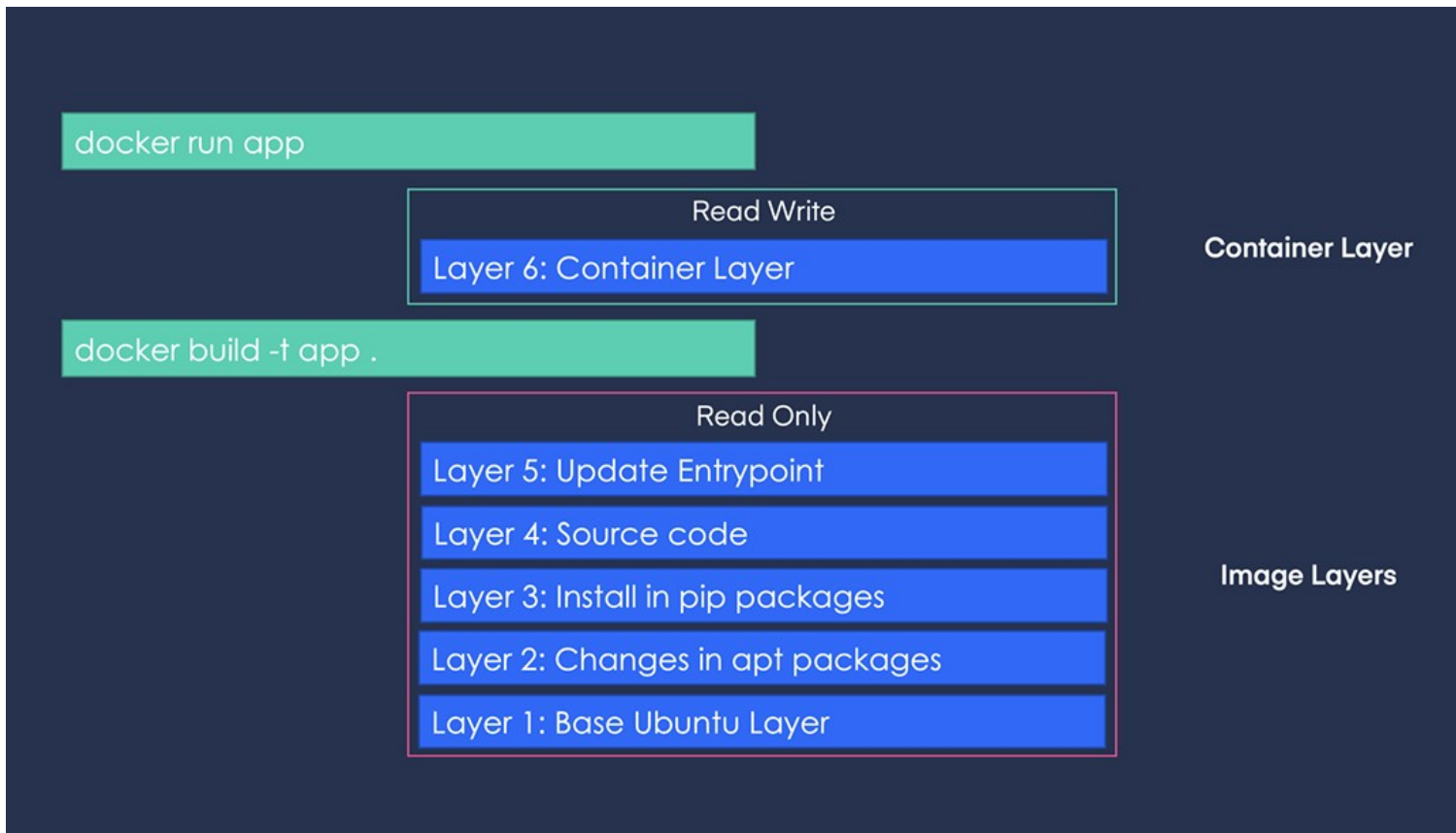
Docker Images

Lab Docker 1



Docker Containers

- Containers are running copies of a Docker image
- Containers have an additional write-able layer added to the image layers



Running Containers

- The **docker run** command starts a container based on an image
- The image contains a default command to run when it starts
 - Once the command completes, the container exits
 - Some containers, running a web server for example, do not exit
 - These have to be shut down with the **docker stop** command or **docker kill**
- Stopped containers are not destroyed but can be restarted
 - The command **docker start** restarts a stopped container but not exited containers
 - The command **docker create** creates a container but does not run it
 - The command **docker run** = **docker create** followed by **docker start**
- Specific commands inside a container can be executed
 - Inside an already running container with **docker exec <cmd>**
 - Or by starting up a container with **docker run <cmd>**

Running Containers

- Running containers have a hash id just like images
 - They also can have an optional name **docker run ubuntu --name zippy**
 - Containers are assigned default names otherwise
- There can be multiple containers created from a single image
- Commands used to work with containers
 - **docker ps** – lists all the running processes related to containers
 - **docker ps -a** – lists all of the running and exited processes related to containers
 - Using **docker container ls (-a)** gives exactly the same output
- Docker keeps logs of all activity in each container
 - We can access both a container's logs and monitor its running processes

Running Containers

- Interactive terminal connections allow us to work within a container if the container supports a shell
 - To work with a shell in Ubuntu we could run **docker run -it ubuntu**
 - If we omit the **-it**, the shell will start up and immediately exit
 - We can override the default command in the image
 - Normally the nginx image starts a web server and does not exit
 - We can start a shell instead with the following command **docker run -it nginx bin/sh**
 - For a running container, we can execute a command using **docker exec -it <container id>**

Running Docker Containers

Demo



Running Docker Containers

Docker Lab 2



Docker Networks

- Docker engine runs a set of private networks
 - Each container gets an IP address on the docker network
 - If the container provides a service, it normally is exposed through a port on the container
 - The docker engine will map ports on the host networks to ports on containers
- Private docker networks
 - Allow containers to run without interfering with IP addresses or ports on the host system
 - Network types:
 - **Bridge**: the default - creates a private internal isolated network for containers
 - **Host**: allows containers to run on the host network – only implemented for Linux hosts
 - **Overlay**: allows containers on different hosts to communicate with each other
- Exposing ports
 - Services offered by container are specified by port numbers which are made available via port exposing
 - Specific ports to be expose can be defined in an image or a container

Docker Networking

Demo



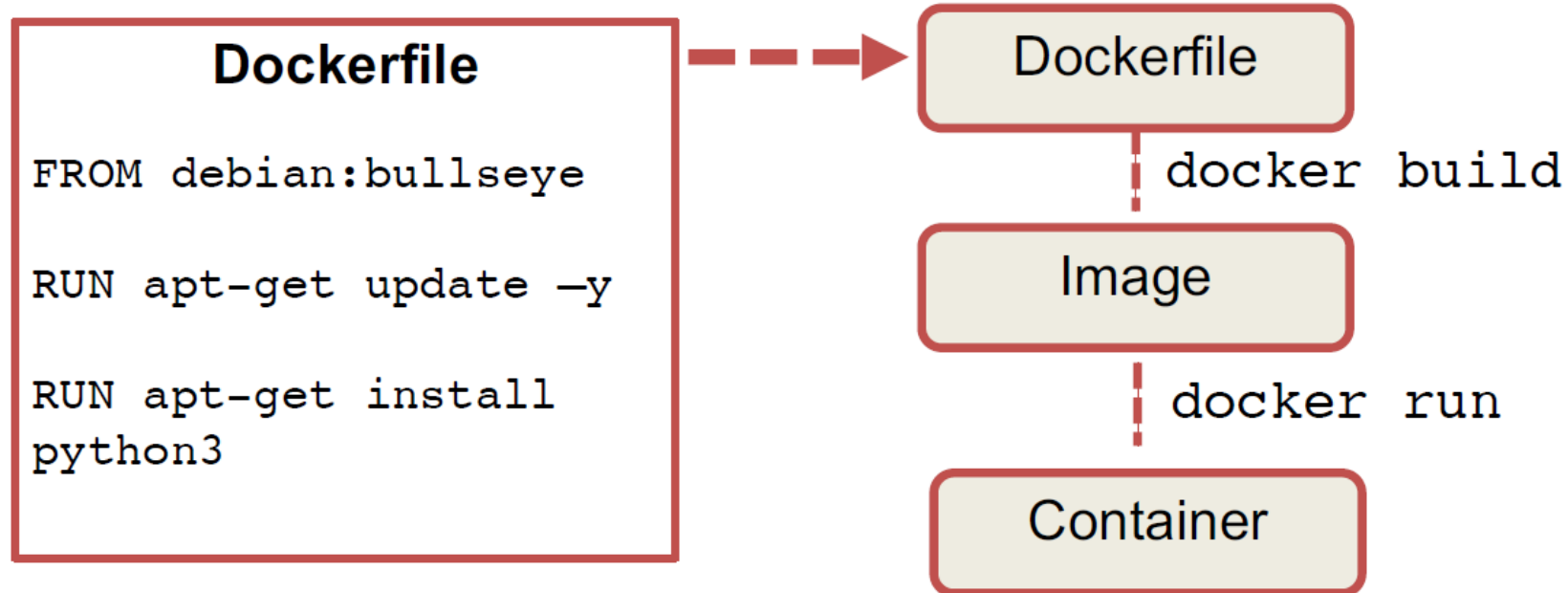
Docker Networking

Docker Lab 3



Building Images with Dockerfile

- Docker can build images automatically by executing instructions in a Dockerfile
 - For example, to build an image with Python three installed on a Debian Linux container



Dockerfile

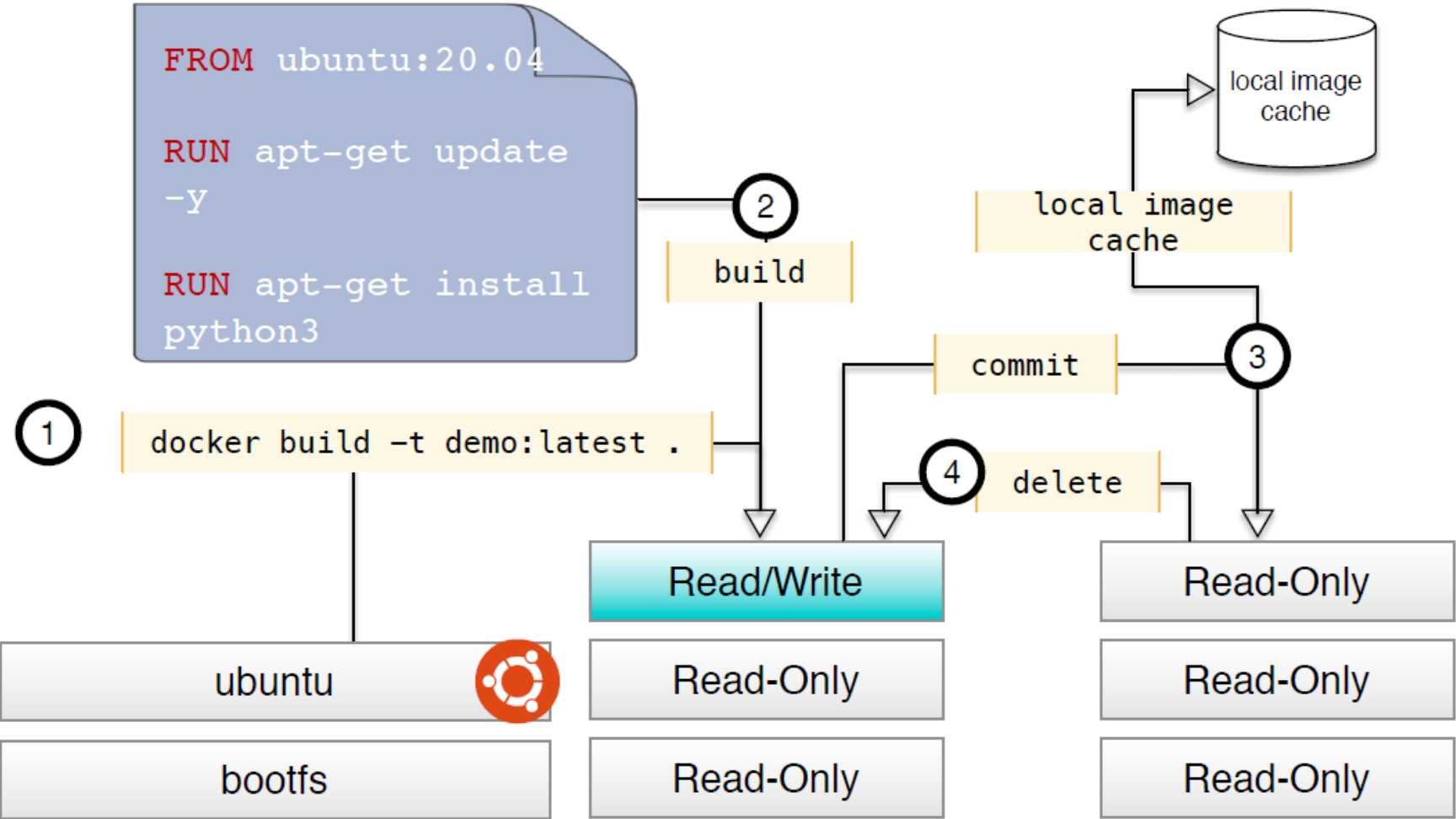
- The Dockerfile is a text file
 - contains the instructions that you would execute on the command line to create an image
 - Docker provides a set of standard instructions to be used in the Dockerfile

Command	Description
#	Comment line
MAINTAINER	Provides name and contact info of image creator
FROM	Tells Docker which base image to build on top of (e.g. centos7)
COPY	Copies a file or directory from the build host into the build container
RUN	Runs a shell command inside the build container
CMD	Provides a default command for the container to run. May be overridden or changed
ADD	Copies new files, directories or remote file URLs
LABEL	Adds metadata to an image

The Build Process

- Docker builds an image by running a series of containers
 - The FROM base image becomes the first layer in the new image
 - The base image is run in a temporary container
 - The first directive, RUN for example, executes and results written to the container's writeable layer
 - The container is committed to a new temporary image
 - The writeable layer now a new image layer
 - This new temporary image is run in a new container, the next directive executes
 - The container is committed to a new temporary image
 - And this continues until the whole Dockerfile is executed

The Build Process



Dockerfile

Demo



Dockerfiles

Docker Lab 4



Docker Monitoring Tools

- Debugging containers can be problematic
- Docker has a number of monitoring tools that can be used
 - `docker logs <container>`: displays console output of the container
 - `docker top <container>`: lists all the processes running in a container
 - `docker stats <container>`: streams real time stats of containers
 - `docker inspect <container>`: displays detailed container configuration

Docker Monitoring Tools

Demo



Docker Volumes

- Volumes are where a directory on the host file system is mounted inside a container
 - Anything written to the volume remains on the host file system
 - Any other container can mount the volume and read what was written
- A wide range of file systems can be mounted
 - Amazon AWS buckets for example
- On Windows, there is not direct access to the underlying file since it is created in the Linux VM
 - However, deleting the volume will delete the underlying file

Docker Volumes

Demo



Docker Volumes

Docker Lab 5



Questions?



Class Project Discussion



End Module

