DevOps - Week 3 - Docker

Muhammad Ali Kahoot
Dice Analytics

QUIZ

- Rebase vs Merge

- How to create & checkout to new branch

- Why and how do conflicts occur

- What is stashing and write its commands

- How to contribute to Open source projects

- Write some best practices of Git

- Write differences between VMs & Containers

- How to persist data in Containers

- Why and how are networks used in Docker

Contribute to Open Source Projects

- Create issue in the main project

- Fork the Project

- Clone your copy of Project

- Create an upstream remote and sync local copy

- Do the work

- Push to your remote repository

- Create a new Pull Request in GitHub

- Review by Maintainers

- Respond to any code review feedback

Best Practices

- Don't commit directly to master
- Create .gitignore file for your projects
- Don't store credentials as code/config in GitHub
- Write meaningful commit message
- Test Your Code Before You Commit
- Use Branches
- Always pull the latest updates
- Protect your project
- The more approvals, the better
- Rebase your branches periodically
- Agree on workflow

DevOps course by M. Ali Kahoot - Dice Analytics
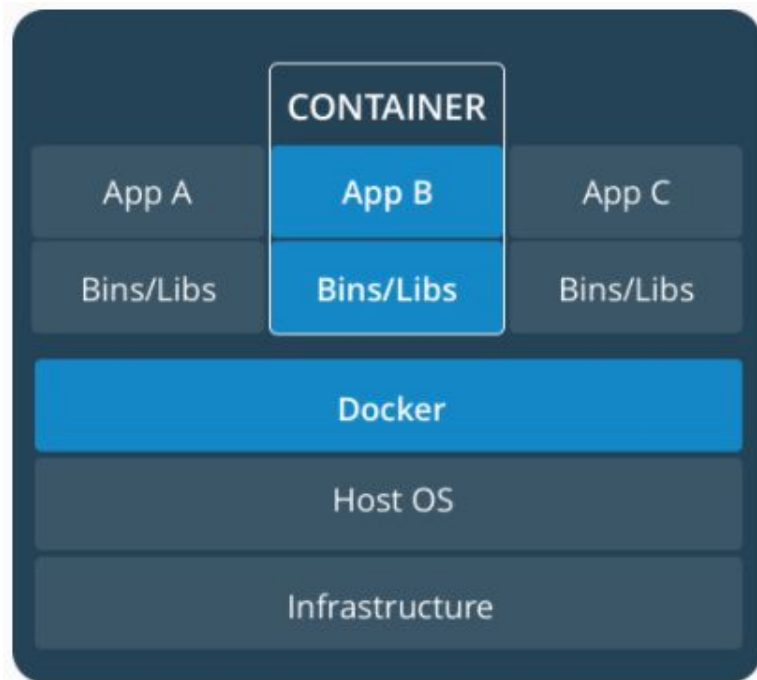
Containers vs VM

CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.
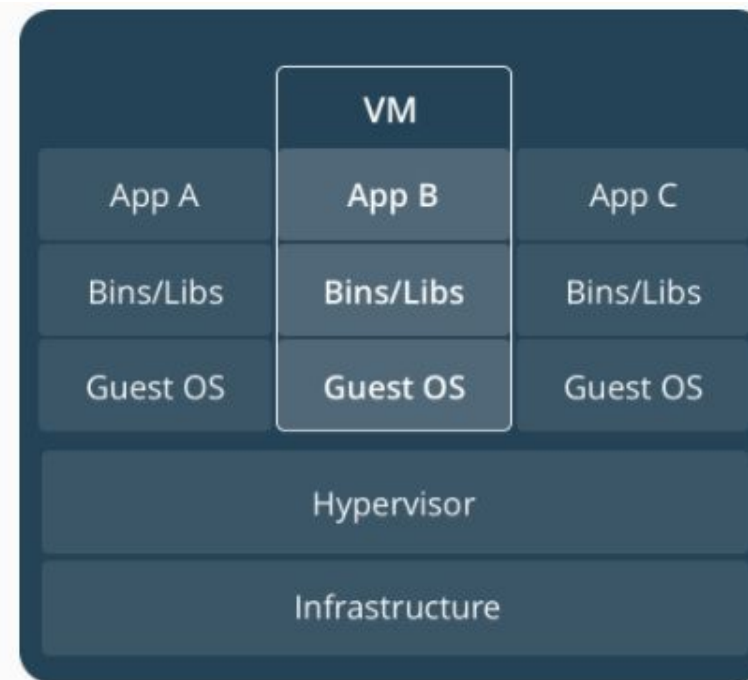
VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

DevOps course by M. Ali Kahoot – Dice Analytics

Containers vs VM

VMs

- Hardware level virtualization i.e. abstraction of physical hardware
- Has own Hardware & OS
- Each VM has a full copy of an operating system + application + binaries + libraries
- can take up to tens of GBs.
- VMs are isolated, apps are not
- Complete OS, Static Compute, Static Memory, High Resource Usage

Containers

- OS level virtualization i.e. abstraction at the app layer (code + dependencies)
- Share hardware, host OS kernel but can have own OS
- take up less space (typically tens to hundreds of MBs in size)
- containers are isolated, so are the apps
- Container Isolation, Shared Kernel, Burstable Compute, Burstable Memory, Low Resource Usage
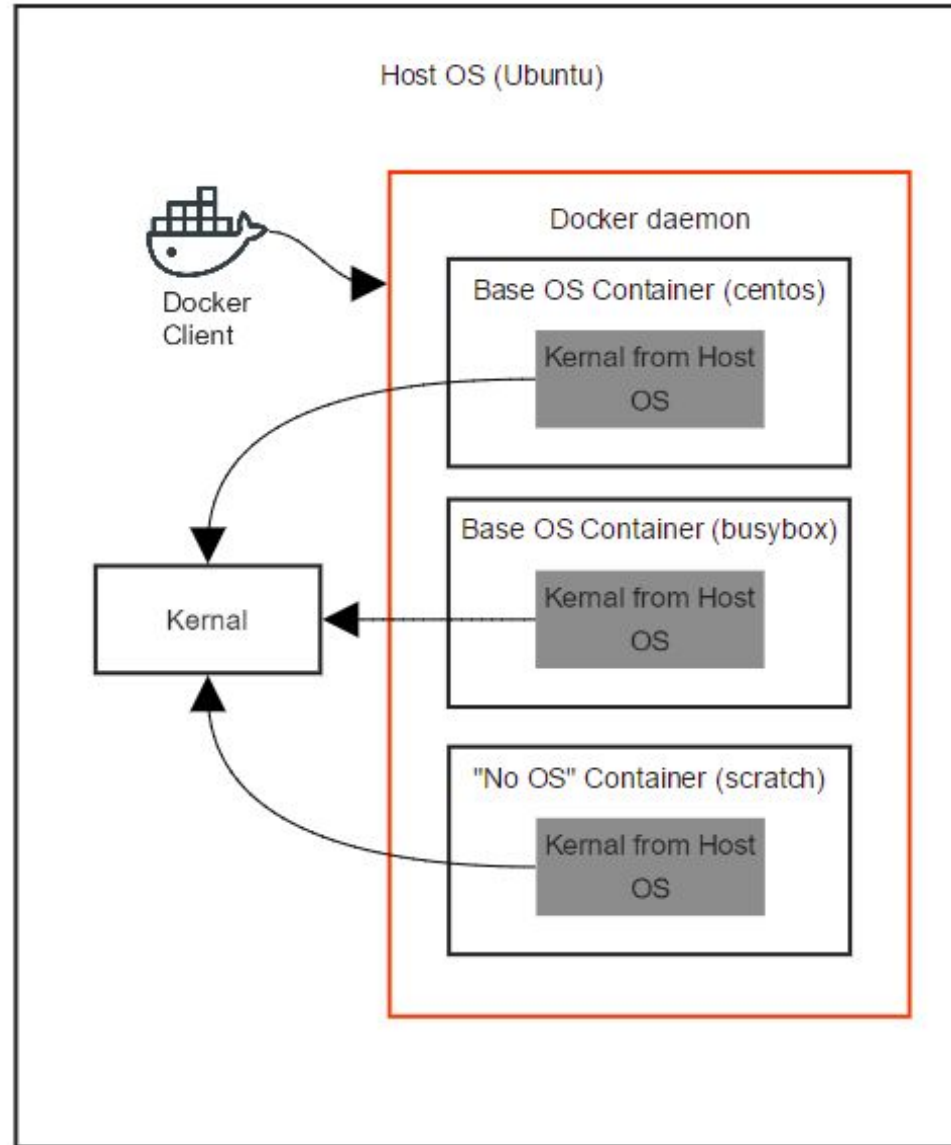
Containers vs VM

VMs

- Ops were responsible for creating VMs, installing Software Dependencies, then installing Software which might not work due to some compatibility issues
- Dev responsible for Software Development and running on local machine vs Ops running the Software on VM with newly installed Libraries
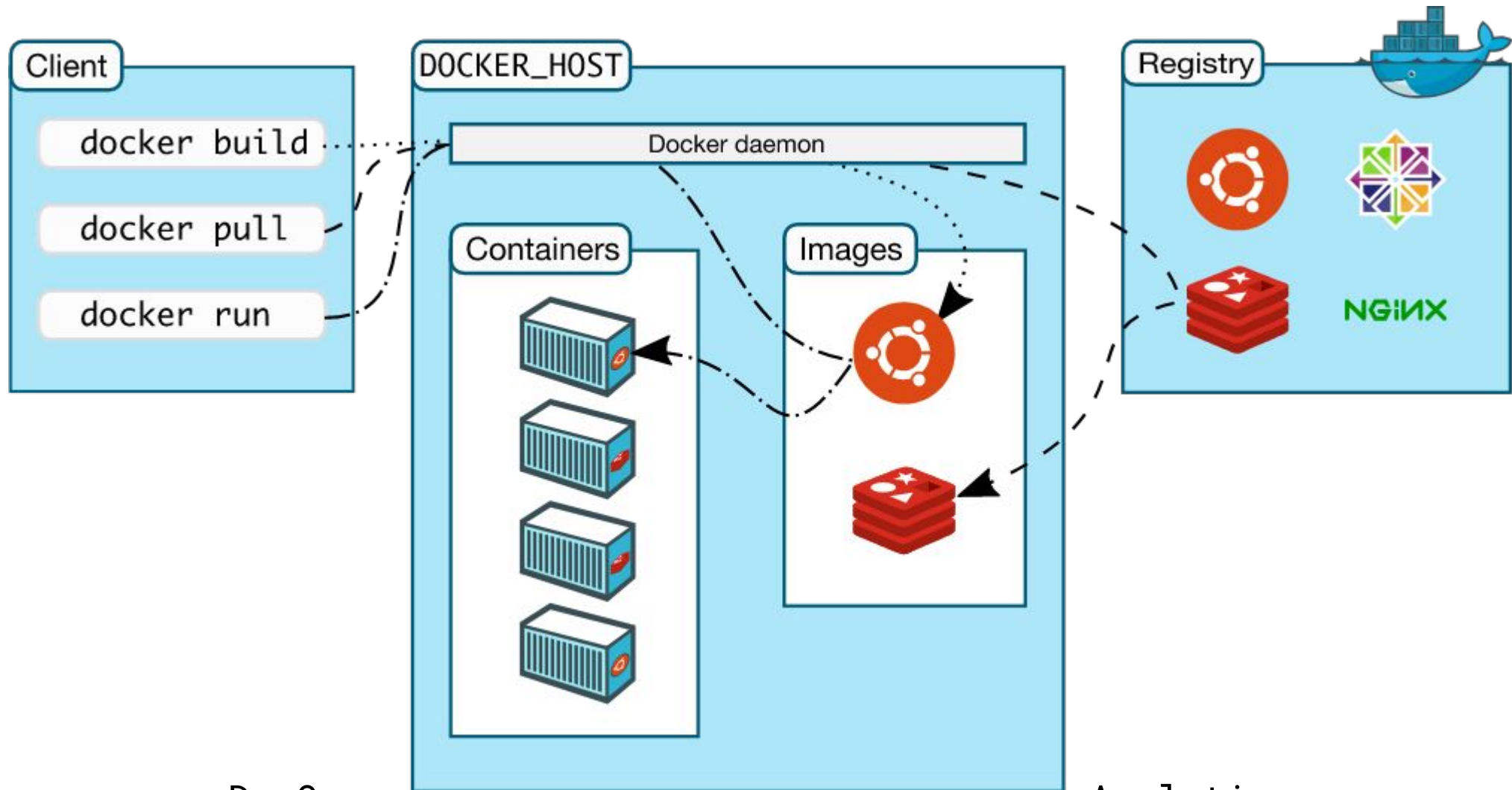- Works on my machine issue

Containers

- Ops are responsible for VM creation and installing Docker only
- Dev writes code and tests in local container based on the same image
- Same image is deployed in Stage, Prod
- Ideally no "WORKS ON MY MACHINE" issue
- Process level isolation but relatively less secure

DevOps course by M. Ali Kahoot - Dice Analytics

Docker Architecture

DevOps course by M. Ali Kanoot - Dice Analytics

Docker Networks

- Each container connected to a private virtual network "bridge"
- Each virtual network routes through NAT firewall on host IP
- All containers on a virtual network can talk to each other
- Best practice is to create a new virtual network for each app
- Can attach containers to more than one virtual network (or none)
- Skip virtual networks and use host IP (--net=host)
- Types of Network Drivers
    - Bridge
    - Overlay
    - Host
    - None

Persistent Data

- Containers are usually immutable and ephemeral

- What about databases, or unique data?

- Docker gives us a feature "persistent data" to resolve this

- Two ways:

  - Volumes: make special location outside of container UFS

  - Bind Mounts: link container path to host path

Persistent Data: Volumes

- Override with docker run -v /path/in/container
- Bypasses Union File System and stores in alt location on host
- Connect to none, one, or multiple containers at once
- By default they only have a unique ID, but you can assign name then it's a "named volume"

DevOps course by M. Ali Kahoot - Dice Analytics

Persistent Data: Bind Mounting

- Maps a host file or directory to a container file or directory

- Basically just two locations pointing to the same file(s)

- Bypasses Union File System

- run -v /Users/stuff:/path/container (mac/linux)

- run -v //c/Users/stuff:/path/container (windows)

DevOps course by M. Ali Kahoot - Dice Analytics

Docker Images

- Images are made up of series of layers and are immutable.

- New layer is built on every application update.

- Can be created from scratch (custom image).

- Can be downloaded via any registry (e.g. Docker Hub).

- Every image extends from a base image. (centos, ubuntu..)

- Every instruction of a Docker file is converted to a layer.

- Images are essentially a snapshot of a container that are then used to base containers upon.

Docker Images

- An image is simply a runnable component with filesystem.

- Docker image is a file composed of multiple layer, used to execute code in docker container. It becomes one or more instances of that containers.

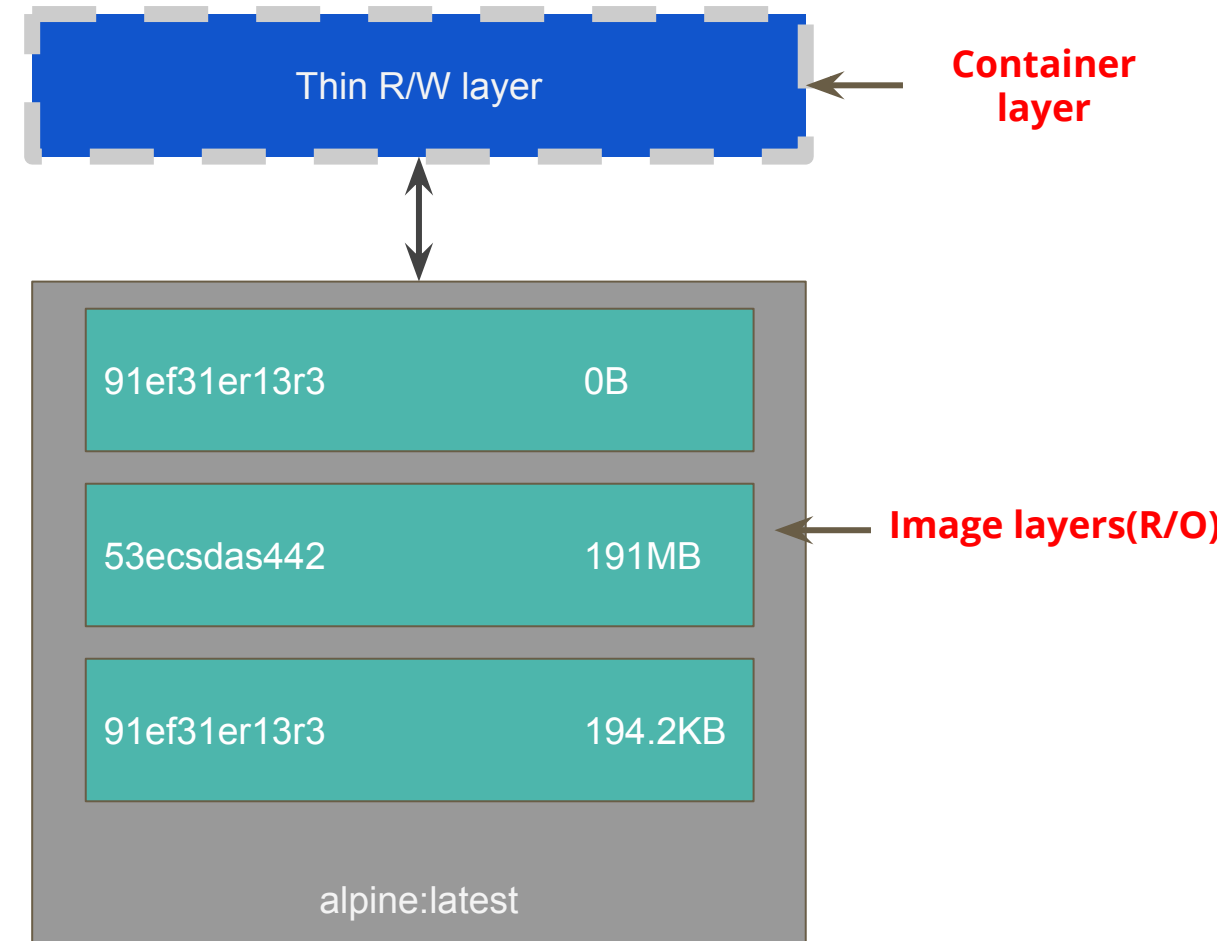- Layers in docker images are just tar files, can be shared b/w images.

Docker Containers

- Runnable instance of an image, created when we start an image with '**run**' command.
- When a container is created a thin read/write layer is put on top of image, Any changes made to the container is put on this read/write layer during the lifetime of the container.
- Can be started, stopped or even deleted (read/write layer is removed) at any time.
- We can start multiple containers of same image and each container runs in complete isolation which means each container maintains its own data safely on the top of the Docker Images as each has its own read/write layer
- You can enter in to container using docker client API.
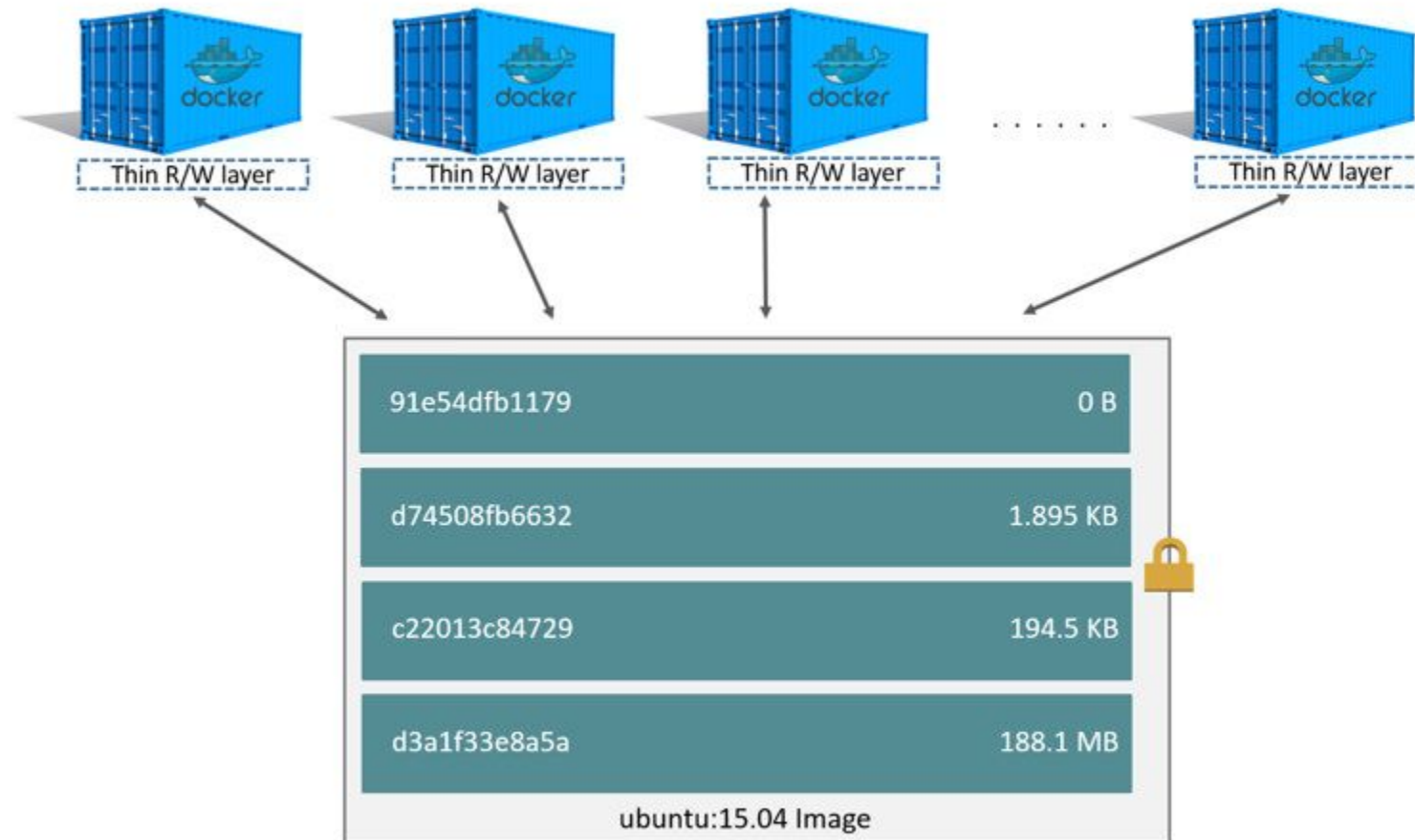
Container vs Image

- Image: A packaged form of an

  application

- Container: Running form of an

  Image

Thin R/W layer

Container layer

91ef31er13r3   0B

53ecsdas442   191MB

Image layers(R/O)

91ef31er13r3   194.2KB

alpine:latest

Multiple Containers from Single Image

Container vs Image

Image

Container

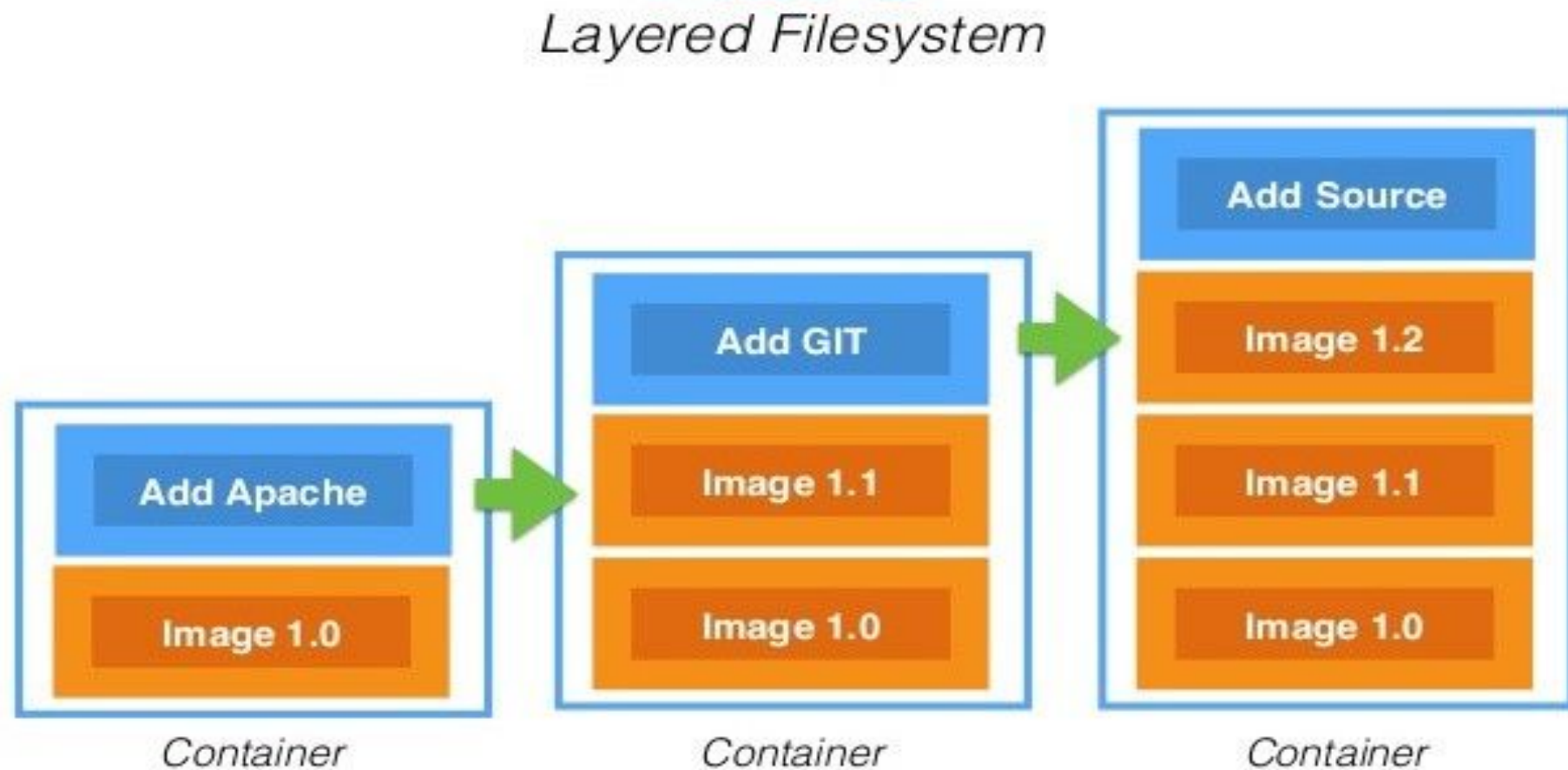DevOps course by M. Ali Kahoot - Dice Analytics

Images based on other Images

Layered Filesystem

Docker Registry

- A central place to Store and Distribute docker images

- Can be publicly or privately hosted.

- Docker images can be pulled/pushed to these registries.

- Some popular hosted registries are

  - Docker Hub

  - AWS ECR

  - Google CR

  - Azure CR

DevOps course by M. Ali Kahoot - Dice Analytics

Docker Hub

- Cloud registry service for sharing application and automating workflows.

- **Features**

  - Unlimited Public and one free Private Docker repositories.

  - Official repositories.

  - Teams and organizations

  - Automated builds

  - Webhooks

Lab

- Create a Docker Hub account.

  ○ https://hub.docker.com/signup

- Sign in using

  ○ **$ docker login**

Dockerfile

- By default docker looks for a file name '**Dockerfile**' with no extension.
- A Dockerfile is text file that has all the instructions needed to build a docker image.
- It has a simple set of commands like CMD, FROM, RUN,ADD, ONBUILD, VOLUME, etc.
- Here's how it works
  - Create a Docker file
  - Build image using Dockerfile
  - Run a container using the image

DevOps course by M. Ali Kahoot - Dice Analytics

Dockerfile Commands

- **FROM:** Defines the base image.

- **RUN:** is an image build step, the state of container after a RUN command will be committed to docker image.
  ○ E.g. RUN apt update && apt upgrade

- **LABEL:** Any Labels/Metadata for the images

- **CMD:** Specifies what command needs to run by default while launching the container, can be overridden at time of container start.

- **ENTRYPOINT:** Specifies commands needed to be run at the start of the container, can't be overridden at the time of starting the container.

DevOps course by M. Ali Kahoot - Dice Analytics

Dockerfile Commands

- **WORKDIR:** Specifies working directory for RUN, CMD, COPY, ADD and ENTRYPOINT instructions.

- **EXPOSE**: exposes the local container port, on which container listens for connections.

- **ENV**: Sets the environment variable <key> to the value <value>

- **ARG**: Defines a variable that users can pass at build-time to the builder with the docker build command, using the --build-arg <varname>=<value> flag

It's a best practice to set ENTRYPOINT as an image's main command and CMD to set default tags.

Dockerfile commands (ADD vs COPY)

ADD

- Lets you copy from multiple sources
  - Remote url
  - Can extract from source to destination directly

COPY

- Lets you copy only from
  - Local file
  - Host machine where you're building images

DevOps course by M. Ali Kahoot - Dice Analytics

Lab: Add vs Copy

https://github.com/kahootali/docker-samples/tree/master/add-vs-copy
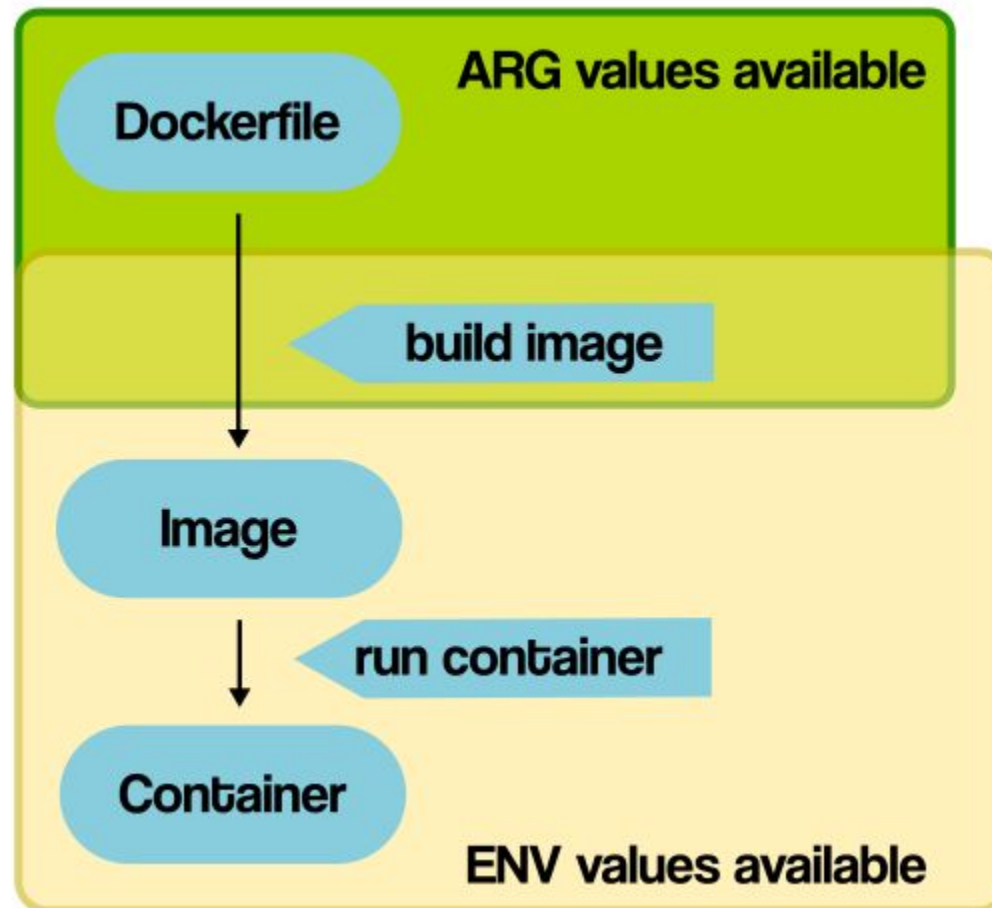
Dockerfile commands (ENV vs ARG)

ENV

- For default values for your future environment variables in your container

- Run time variable

- If not defined, so can be defined while running as well

- Used while docker run -e NAME=Ali Kahoot

ARG

- For default values for variables during image build

- Build time variable

- A running container won't have access to an ARG variable value

- Used by docker build -build-arg NAME=Ali Kahoot

DevOps course by M. Ali Kahoot - Dice Analytics

ENV vs ARG

DevOps course by M. Ali Kanoot – Dice Analytics

Lab: ENV vs ARG

https://github.com/kahootali/docker-samples/tree/master/arg-vs-env

Dockerfile commands (ENTRYPOINT vs CMD vs RUN)

- RUN: Build time command, if you want to run any command during the build of the image

- Entrypoint & CMD are runtime commands that are used to specify which command needs to run while running the container

- ENTRYPOINT: The executable to run while running the container

- CMD: If Entrypoint is not defined, CMD becomes the 1st command but if defined, CMD is passed as an argument to Entrypoint

Dockerfile commands (ENTRYPOINT vs CMD)

If you want to run a command e.g. **echo Hello Ali Kahoot** for your ubuntu image.

- Entrypoint: "echo" CMD: 'Hello Ali Kahoot'
- Entrypoint: " CMD: echo Hello Ali Kahoot
- Entrypoint: echo Hello Ali Kahoot' CMD: "

Lab: ENTRYPOINT vs CMD

https://github.com/kahootali/docker-samples/tree/master/entrypoint-vs-cmd

Example DockerFile

Dockerfile

```
FROM ubuntu:latest

COPY . /app

RUN make /app

CMD python /app/app.py
```

Layers

From creates a layer from ubuntu docker image.

COPY add files from your Docker client's current directory.

RUN runs the command specified, here make builds your application with make

CMD specifies what command to be run within the container.

Building custom Docker images

- Docker build command builds the docker image with specified tag using instructions defined in the Dockerfile and use local cache if image already exists.

Usage: $ docker build –t <yourImageTagName> <PathToDockerFile>

Best Practices for creating an image

- Make the size of the image as low as possible

- Make the build time of the image as less as possible

- Make the image as secure as possible

Best Practices for creating an image

- Avoid including unnecessary files e.g copy all files
- Use .dockerignore
- Don't install unnecessary packages
- Decouple applications
- Use official images when possible
- Minimize the number of layers
- Sort multi-line arguments
- Leverage build cache
- Use multi-stage builds

Lab - Best Practices for Dockerfile

- Create a simple python web server.

- Create a docker file

- Build an image from Dockerfile

- Check newly created image

- Run your container

- Test Application

Lab - Best Practices for Dockerfile - Leverage Cache

- Create a simple python web server.
- Create a docker file

These files are present at

https://github.com/kahootali/docker-samples/tree/master/bad-practices-images/cached-layers

Lab - Best Practices for Dockerfile - Leverage Cache

- Build an image from Dockerfile

```
docker build -t diceanalyticsweek3lab1 .
```

- Check newly created image

```
docker images
```

- Run your container

```
docker run -d --name diceanaylticsweek3lab -p 8082:8000 diceanaylticsweek3lab1
```

- Test Application

Open localhost:8082 on your browser to test your application.

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Best Practices for Dockerfile - MultiStage Build

These files are present at

https://github.com/kahootali/docker-samples/tree/master/bad-practices-images/multi-stage-builds

Docker Tags

- Conveys useful information about specific image version/variant.

- We can either assign tag during an image build

```
docker build -t kahootali/myImage:1.0 .
```

, or can explicitly use **'tag'** command

```
docker tag existingImage kahootali/myImage:1.0
```

- Alias to ID of your image, which helps easy identification of images.

Build Images from Container

- Really handy when you're working out how an image should be constructed.

- You want an image, but not sure of the commands of how to build it

- You can just keep tweaking the container until it works like you want.

- docker container commit command is used for this purpose.

```
docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

Lab - Docker Images

- Run container and make changes

- Build image from a container.

- Tag newly created image.

- Login to your Docker Hub account.

- Push image to Docker Hub.

Lab - Docker Images

- Run container and make changes

```
docker run -it ubuntu bash

cat <<EOF >> /home/file.txt

Name: Muhammad Ali Kahoot

Container: Diceanalyticslab

EOF
```
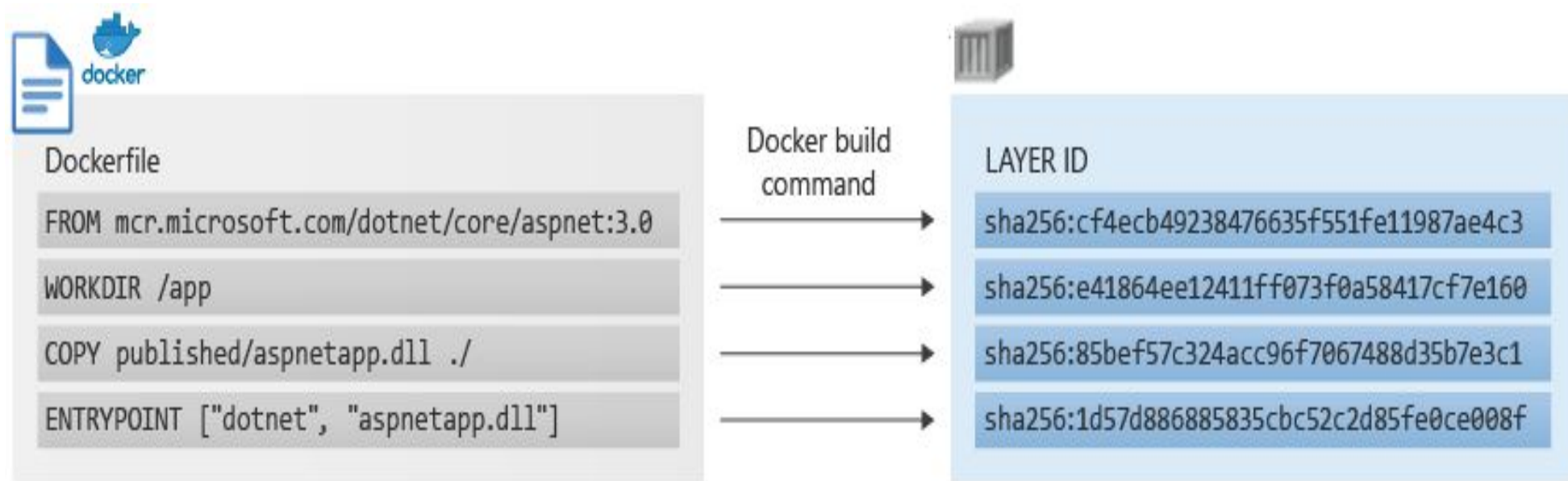
- Build image from a container

```
docker commit  <CONTAINER_ID>  image-from-container
```

Lab - Docker Images

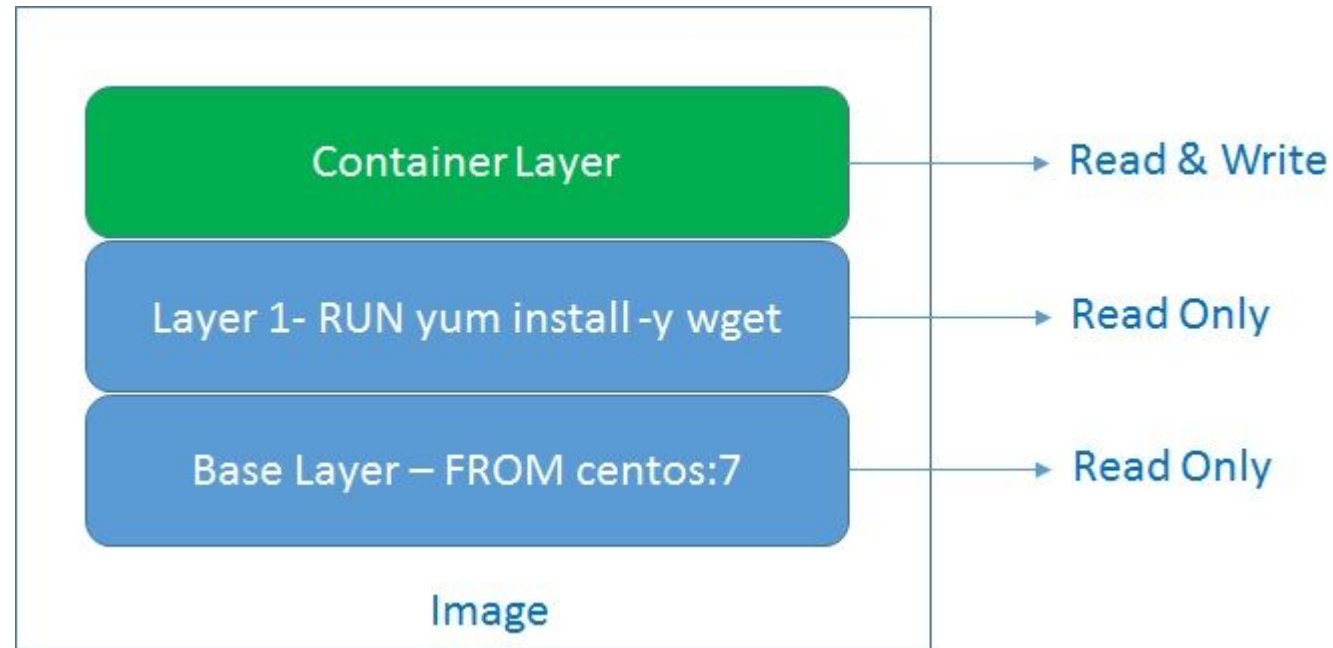- Tag newly created image
  ```
  docker tag <IMAGE_ID> <yourDockerHubUsername>/<repoName>:<TAG>

  E.g docker tag 4ffda6c9a8e3 kahootali/diceanalytics:v1.0
  ```

- Login to your Docker Hub account
  ```
  docker login
  ```

- Push image to Docker Hub
  ```
  docker push <yourDockerHubUsername>/<repoName>:<TAG>
  docker push kahootali/diceanalytics:v1.0
  ```

Layers for Image

Layers for Image

Container Layer → Read & Write

Layer 1- RUN yum install -y wget → Read Only

Base Layer – FROM centos:7 → Read Only

Image

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Applications

We have an application written in different languages which we will dockerize. Almost all of them provide 2 endpoints

- /hello: Prints Hello World

- /count: Prints count of the endpoint visited

We have a sample repo with this applications in Dockerized form and we will be running them so first star & then fork the Repo

https://github.com/kahootali/docker-samples

Lab - Dockerizing Applications

- Dockerize Golang App

- Dockerize Java App

- Dockerize Node App

- Dockerize Node Express App

- Dockerize Python App

Lab - Dockerizing Go Application

- Dockerize Golang App

https://github.com/kahootali/docker-samples/tree/master/golang-app

Lab - Dockerizing Go Application

```dockerfile
FROM golang:alpine as build
WORKDIR /app
COPY main.go ./
RUN go env -w GO111MODULE=auto
RUN go build main.go

FROM alpine
LABEL name="Golang Application" \
      maintainer="Ali Kahoot <kahoot.ali@outlook.com>" \
      summary="A Golang Sample application"
EXPOSE 8080
WORKDIR /app
COPY --from=build ./app/main ./
CMD ["./app/main"]
```

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Go Application

```
docker build -t golang-app .
docker run -it --rm --name golang-app --init -p 8080:8080 golang-app
```

Go to browser and access

```
localhost:8080/hello
localhost:8080/count
```

Now press CTRL+C to exit the container, Now we will try to override the CMD command.

Lab - Dockerizing Go Application

```
docker run -it --rm --name golang-app --init -p 8080:8080 golang-app sh
```

Now you will see that the golang app didn't start rather a shell has started and you are attached to it. Now you can run any command that a shell can run. Now we will run the golang app from the shell. Run

```
./app
```

Access the browser again and application should be working. Now press CTRL+C to stop the application, But the container will still be running. Now press exit to exit from the container.

Lab - Dockerizing Java Application

- Dockerize Java App

https://github.com/kahootali/docker-samples/tree/master/java-app

Lab - Dockerizing Java Application

```
FROM maven:3.8.1 as build
COPY src /usr/src/app/src
COPY pom.xml /usr/src/app
RUN mvn -f /usr/src/app/pom.xml clean package -Dmaven.test.skip=true

FROM gcr.io/distroless/java:11
LABEL name="Java Application" \
      maintainer="Ali Kahoot <kahoot.ali@outlook.com>" \
      summary="A Java Spring Boot application"
WORKDIR /app
EXPOSE 8080
COPY --from=build /usr/src/app/target/*.jar artifacts/app.jar
CMD ["artifacts/app.jar"]
```

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Java Application

```
docker build -t java-app .
docker run -it --rm --name java-app --init -p 8081:8080 java-app
```

Go to browser and access

```
localhost:8081/hello
localhost:8081/count
```

We are using distroless image here, so it will not have a shell available so you can't exec into it, as the purpose is to have very small and more secure image.

ENTRYPOINT vs CMD

As we are using the distroless image of Google so its ENTRYPOINT command is
```
java -jar
```

And in CMD we are passing ["artifacts/app.jar"] so it will be passed as an argument to ENTRYPOINT command. So thats how your application will run as the complete command will be
```
java -jar ["artifacts/app.jar"]
```

ENTRYPOINT vs CMD

Now we will try to override the ENTRYPOINT & CMD and run again

```
docker run -it --rm --name java-app --init --entrypoint java -p 8081:8080
java-app -jar artifacts/app.jar
```

Now the application has started again but you can see this time -jar artifacts/app.jar(CMD) is passed as argument to java(ENTRYPOINT) and full command again becomes

```
java -jar ["artifacts/app.jar"]
```

Lab - Dockerizing Python Application

- Dockerize Python App

https://github.com/kahootali/docker-samples/tree/master/python-app

Lab - Dockerizing Python Application

```dockerfile
FROM python:3.6-alpine
LABEL name="Python Application" \
      maintainer="Ali Kahoot <kahoot.ali@outlook.com>" \
      summary="A Sample Python application"
WORKDIR /app
EXPOSE 8080
RUN pip install flask
COPY app.py ./
CMD [ "python", "./app.py" ]
```

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Python Application

```
docker build -t python-app .
docker run -it --rm --name python-app --init -p 8082:8080 python-app
```

Go to browser and access

```
localhost:8082/hello
localhost:8082/count
```

Lab - Dockerizing Node Application

- Dockerize Node App

https://github.com/kahootali/docker-samples/tree/master/node-app

Lab - Dockerizing Node Application

```dockerfile
FROM node:10-alpine
LABEL name="Node Application" \
      maintainer="Ali Kahoot <kahoot.ali@outlook.com>" \
      summary="A Node Sample application"
# Create app directory
WORKDIR /app
EXPOSE 8080
COPY index.js ./
CMD [ "node", "index.js" ]
```

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Node Application

```
docker build -t node-app .
docker run -it --rm --name node-app --init -p 8083:8080 node-app
```

Go to browser and access

```
localhost:8083/
```

Lab - Dockerizing Node with Express Application

- Dockerize Node Express App

https://github.com/kahootali/docker-samples/tree/master/node-express-app

Lab - Dockerizing Node Application

```
FROM node:10-alpine

LABEL name="Node Express Application" \
      maintainer="Ali Kahoot <kahoot.ali@outlook.com>" \
      summary="A Node Express application"

WORKDIR /app

EXPOSE 8080

COPY package*.json ./

RUN npm install

COPY server.js ./

CMD [ "npm", "start" ]
```

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerizing Node Application

```
docker build -t node-express-app .
docker run -it --rm --name node-express-app --init -p 8084:8080
```

```
node-express-app
```

Go to browser and access

```
localhost:8084/hello
```

```
localhost:8084/count
```

Dockerizing Microservices

DevOps course by M. Ali Kahoot - Dice Analytics
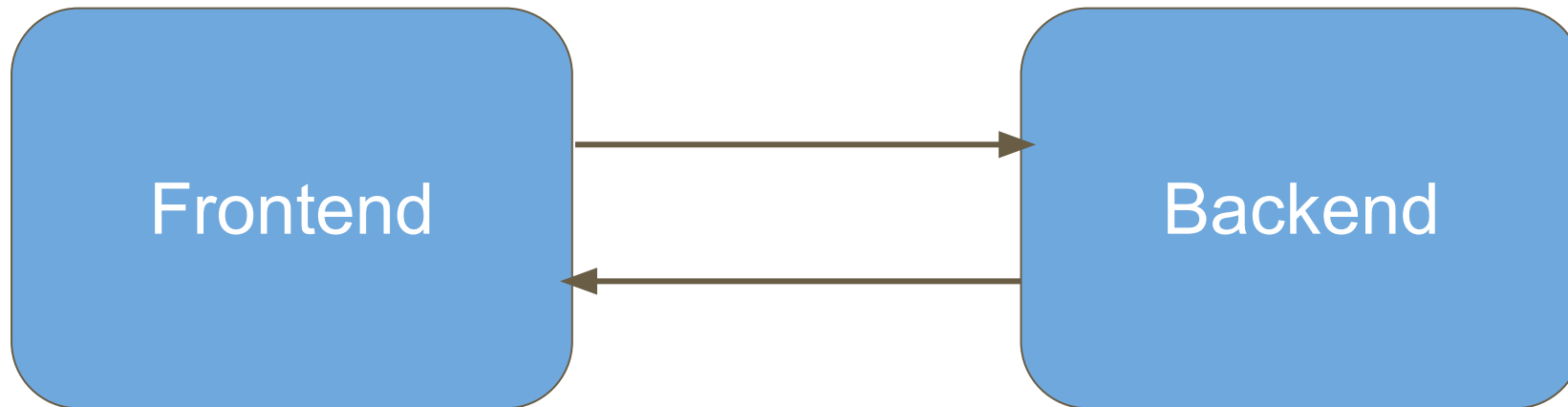
Dockerizing Microservices

Frontend → Backend

/hello
/instructor/id: Hello and call backend/ instructor/id
/student/id: Hello and call backend/student/id

/hello
/instructors
/instructor/id
/students
/student/id

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Running Dockerized Microservices

- Dockerize Python Backend App

- Dockerize Node Frontend App

- Communicate with each other

- Create Network & Deploy apps in network

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Running Dockerized Microservices

- Dockerize Python Backend App

https://github.com/kahootali/docker-samples/tree/master/microservices/backend-python

- Dockerize Node Frontend App

https://github.com/kahootali/docker-samples/tree/master/microservices/frontend-node

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Dockerize Python Backend App

```
cd ../microservices/backend-python/

docker build -t python-backend .

docker run -it --rm --name backend --init -p 9090:8080 python-backend
```

Go to browser and access

```
localhost:9090/hello

localhost:9090/instructors

localhost:9090/instructor/1 or 2

localhost:9090/students

localhost:9090/student/{id}
```

Now keep the backend running and we will try to run the frontend in other terminal that calls the backend through API.

Lab - Dockerize Node Frontend App

Move to microservices/frontend-node folder. Check the server.js file

```
cd ../frontend-node/

docker build -t frontend-app .

docker run -it --rm --name frontend --init -p 9091:8080 frontend-app
```

Go to browser and access

```
localhost:9091/hello

localhost:9091/instructor/{id}

localhost:9091/student/{id}
```

This will not work. Check the code of Frontend trying to call the backend and see the issue

Lab - Dockerize Node Frontend App

As containers cannot access your machines loca applications. Press Ctrl + C. Lets connect to container's shell and try curl.

```
docker run -it --rm --name frontend --init -p 9091:8080 frontend-app sh
curl localhost:9090/instructors
```

But if you check the browser on localhost:9090/hello, the backend app is working. So one container cannot access other containers and host's other processes by default.

DevOps course by M. Ali Kahoot - Dice Analytics

Lab - Creating Network and attaching applications

Exit from previous containers. Create a new Docker Network.

```
docker network create application
```

Run backend again and attach to network

```
docker run -it --rm --name backend --network application --init -p
9090:8080 python-backend
```

Change frontend-app's code line 20 and line 33 Change **localhost:9090** to **backend:8080**, the url will become
http://backend:8080/instructor/${id}}

Lab - Creating Network and attaching applications

Now build the image again(You should know the command by now, if not see above), and Run Frontend application and join network.

```
docker run -it --rm --name frontend --network application --init -p
9091:8080 frontend-app
```

Now access in browser

```
localhost:9091/hello
localhost:9091/instructor/{id}    id=1/2
localhost:9091/student{id}        id=1/2/3/4/5
```

DevOps course by M. Ali Kahoot - Dice Analytics