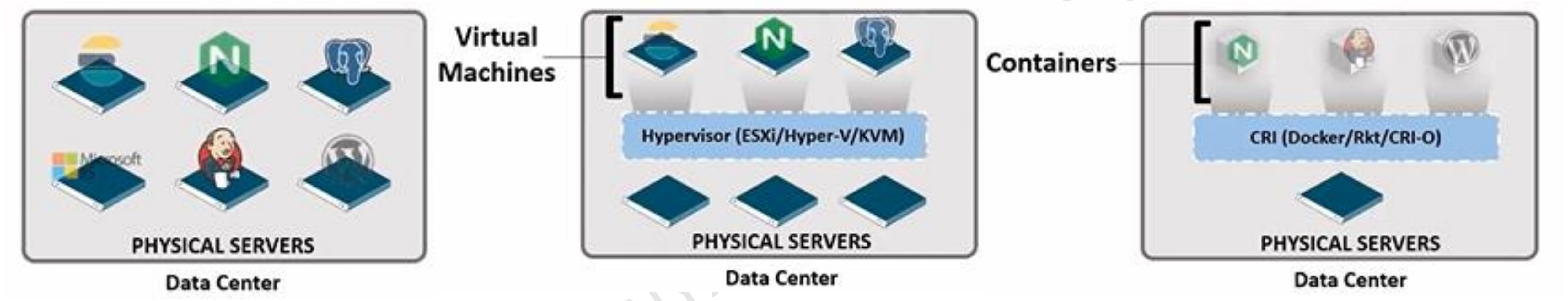
Docker Fundamentals

Agenda

- Introduction to containers
- Docker Fundamentals
- Docker Architecture
- Lab – Docker Set Up
- Docker Images
- Docker Containers
- Container Network Model
- Docker Volumes
- Lab on Build and Ship Images

Introduction

Overview of Containers

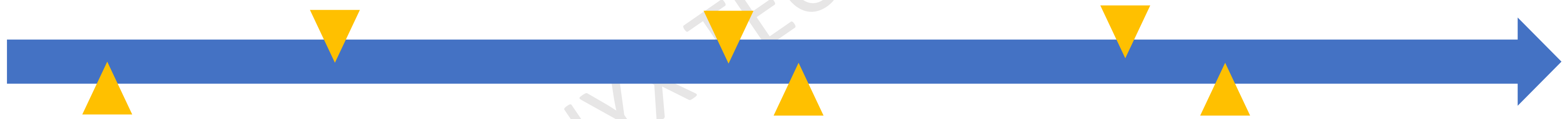


- Huge Infrastructure cost
- Poor Resource utilization
- Dedicated Infrastructure teams
- Long Downtime

- On the fly resource addition like memory, CPU, network
- Better resource utilization
- Less Downtime
- Need dedicated teams
- Huge infrastructure cost

- Immutable and Cost effective
- Contains all dependencies
- Easy to deploy
- Build once, run anywhere
- Single DevOps team
- Speedy recovery

Origins of Docker Project



2004
Solaris Containers / Zones technology introduced

2008
Linux containers (LXC 1.0) introduced

2013
Solomon Hykes starts Docker as an internal project within dotCloud

Mar 2013
Docker released to open source

Feb 2016
Docker introduces first commercial product – now called Docker Enterprise Edition

Today
Open source community includes:
- 3,300+ contributors
- 43,000+ stars
- 12,000+ forks

Results

Speed

- No OS to boot = applications online in seconds

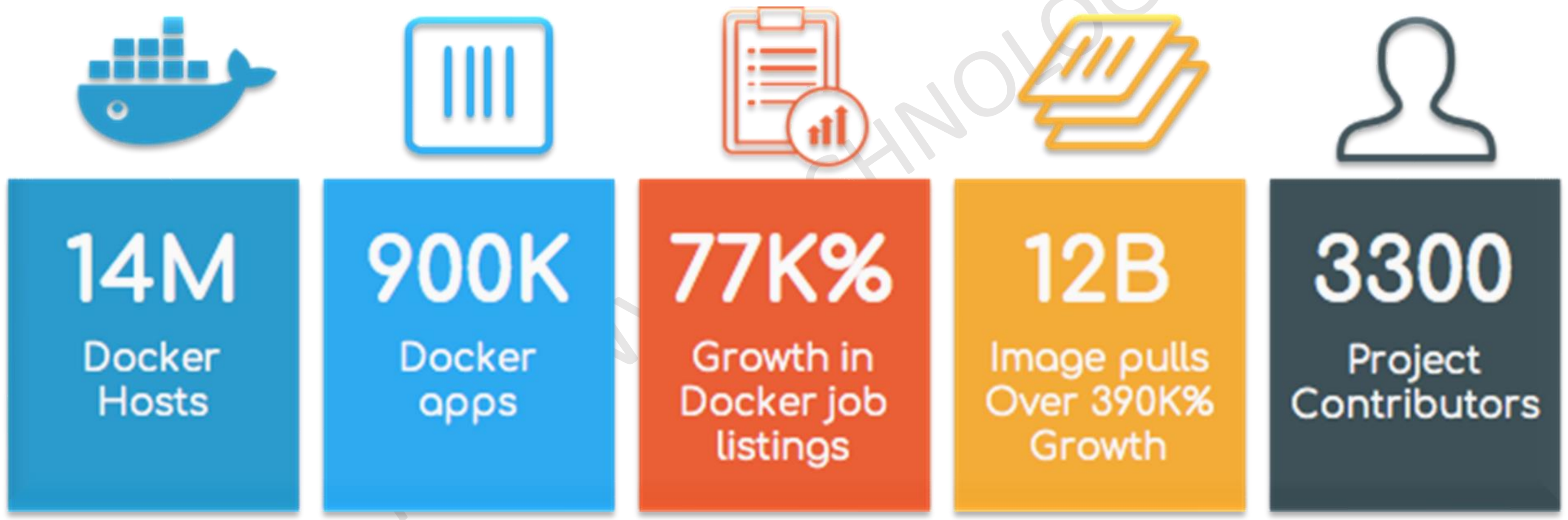
Portability

- Less dependencies between process layers = ability to move between infrastructure

Efficiency

- Less OS overhead
- Improved VM density

Adoption in Just 4 years



14M
Docker Hosts

900K
Docker apps

77K%
Growth in Docker job listings

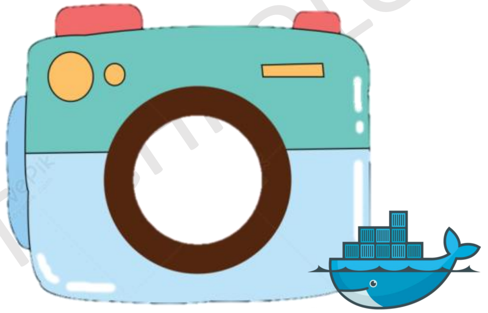
12B
Image pulls Over 390K% Growth

3300
Project Contributors

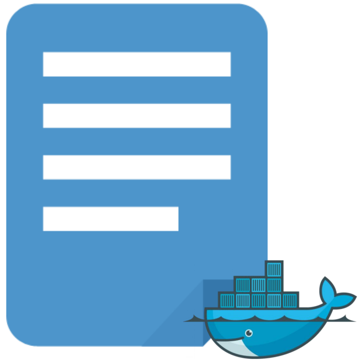
Docker

Docker is a platform to build, ship and run containerized applications.

Container



Image



Dockerfile

Docker Components

Docker Images

- A Docker image is a read-only template with instructions for creating a Docker container.
- It is a template to describe how to build an image.
- For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others.
- A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.
- Docker images are the build component of Docker.

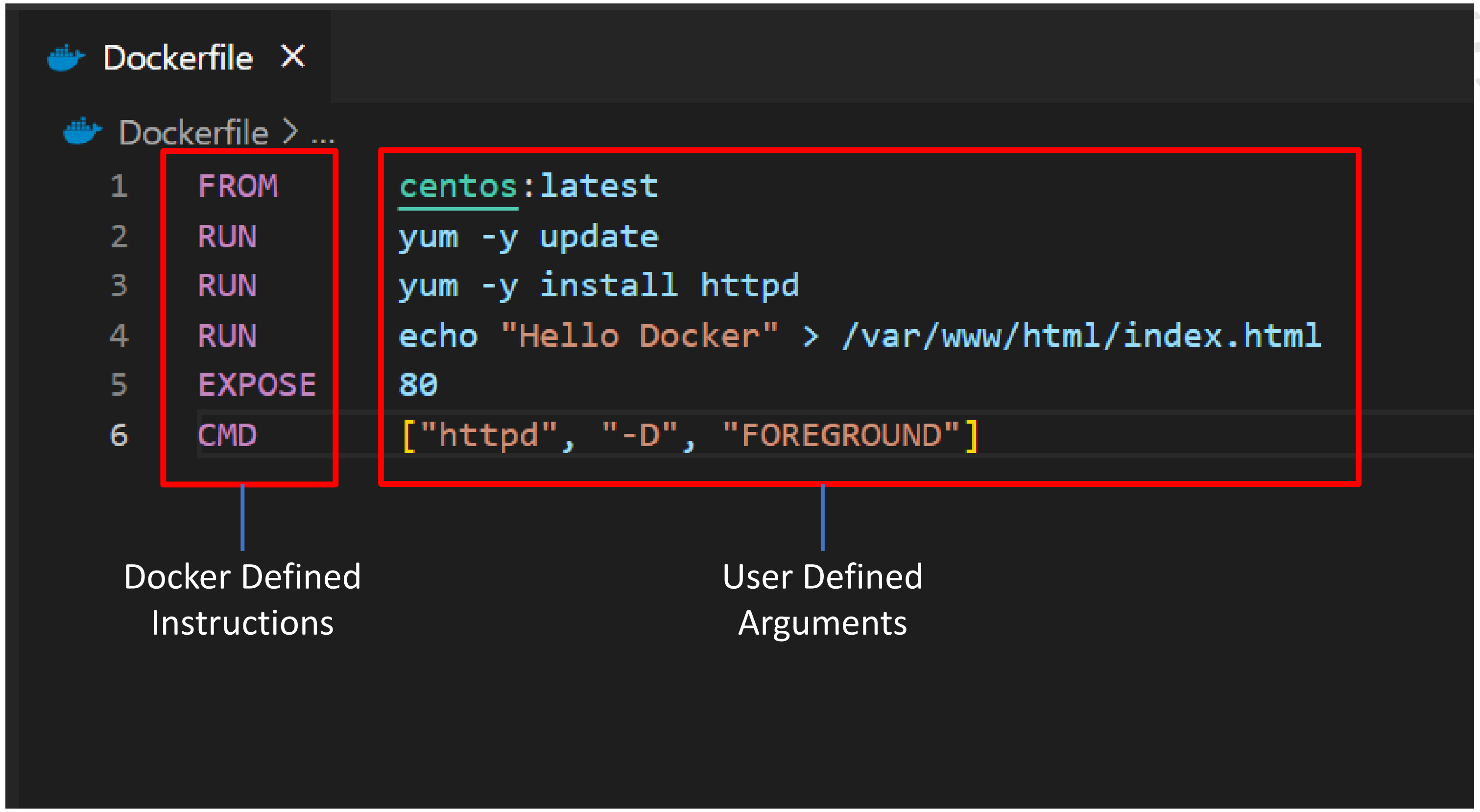
Docker Containers

- A Docker container is a running instance of a Docker image.
- You can run, start, stop, move, or delete a container using Docker API or CLI commands.
- When you run a container, you can provide configuration metadata such as networking information or environment variables.
- Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.
- Docker containers are the run component of Docker.

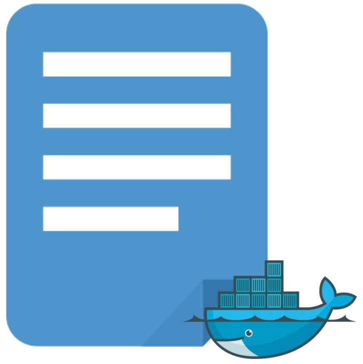
Docker Registries

- A docker registry is a library of images.
- A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
- Docker registries are the distribution component of Docker.
- “Docker Hub” is known as global registry.

INSTRUCTION arguments



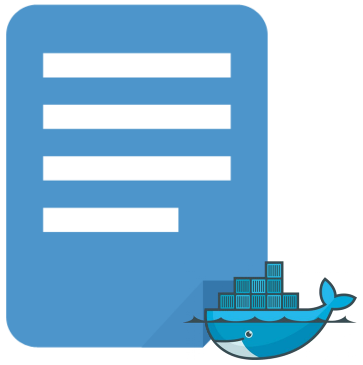
```
FROM    centos:latest
RUN     yum -y update
RUN     yum -y install httpd
RUN     echo "Hello Docker" > /var/www/html/index.html
EXPOSE  80
CMD     ["httpd", "-D", "FOREGROUND"]
```



Dockerfile

Docker Instructions

- FROM
- RUN
- CMD
- LABEL
- MAINTAINER ✖
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- WORKDIR
- ARG
- ONBUILD
- STOPSIGNAL
- HEALTHCHECK
- SHELL


Dockerfile

Docker Arguments

```
FROM ubuntu:latest

COPY test.txt /mydir/
```

Dockerfile

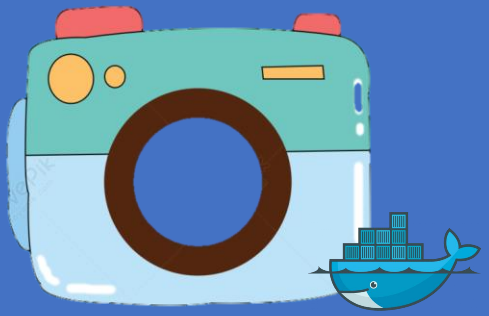
```
CMD ["executable","param1","param2"] (exec form, this is the preferred form)
CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
CMD command param1 param2 (shell form)
```

BUILD ONCE, RUN ANYWHERE

BUILD

Dockerfile

SHIP



Image

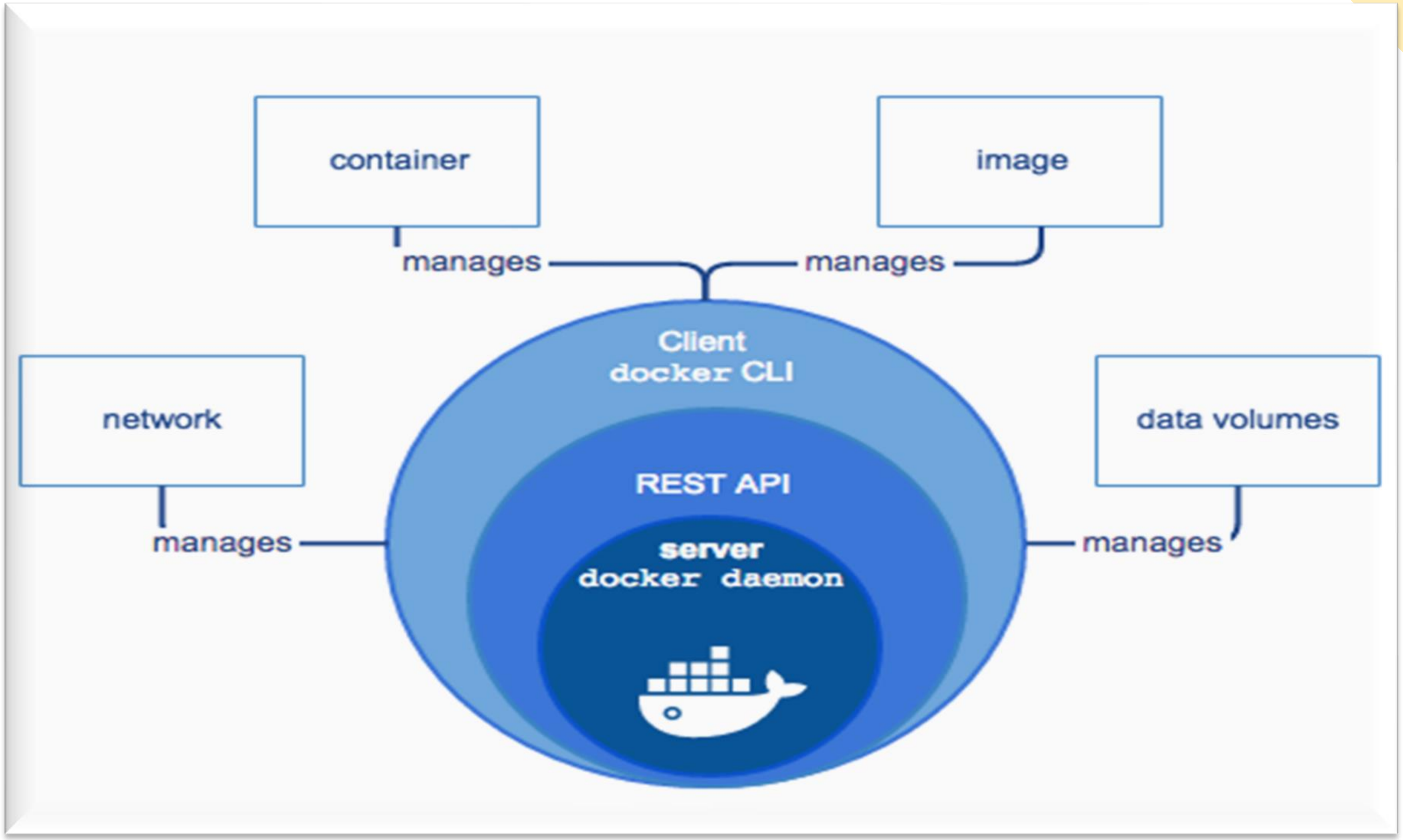
RUN

Container

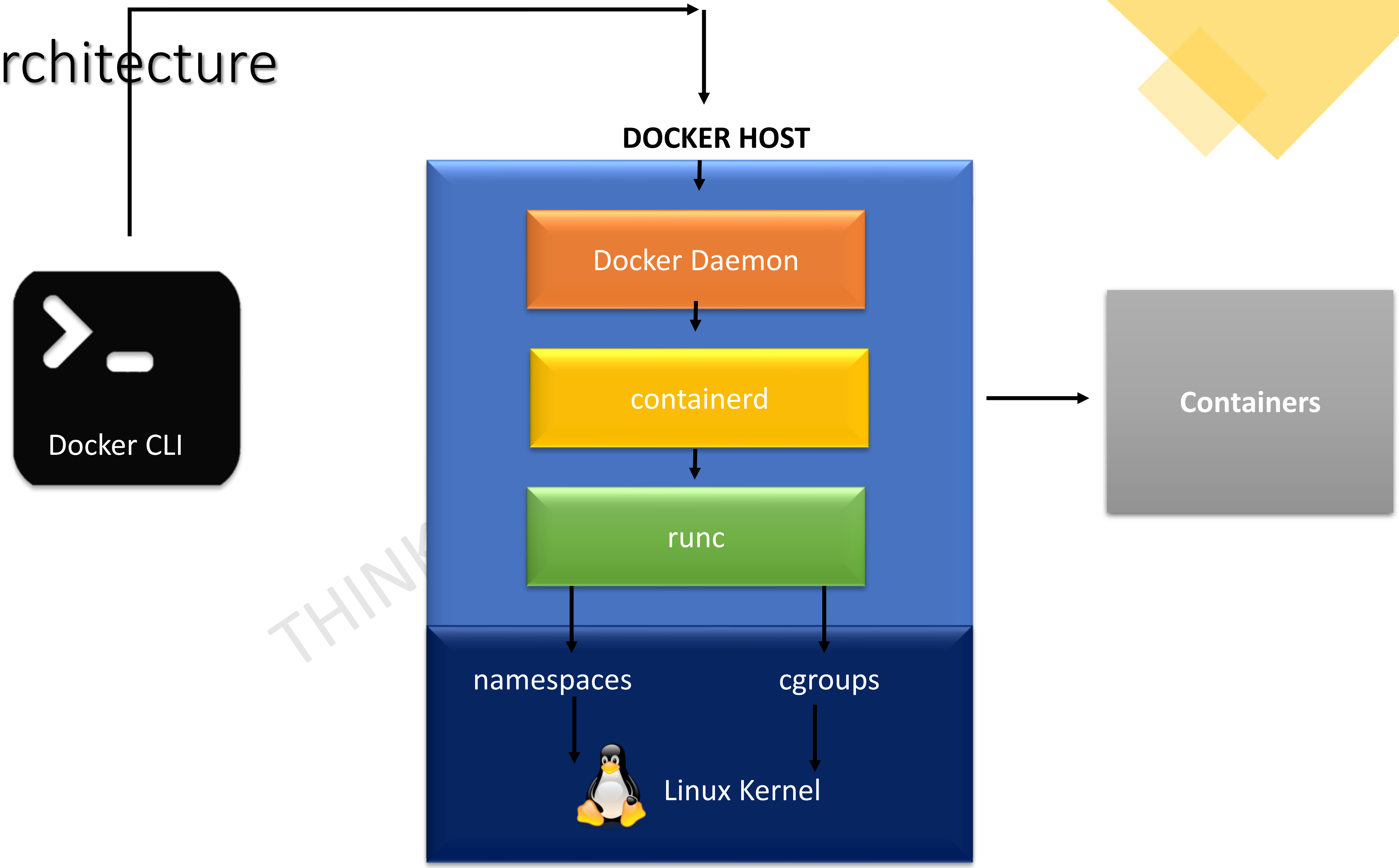
Docker Overview

- 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.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By using Docker's methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

Docker Engine



Docker Architecture



Docker Architecture

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Architecture

- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Docker Architecture

Image
The basis of a Docker container. The content at rest.

Container
The image when it is ‘running.’ The standard unit for app service

Engine
The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry
Stores, distributes and manages Docker images

Control Plane
Management plane for container and cluster orchestration

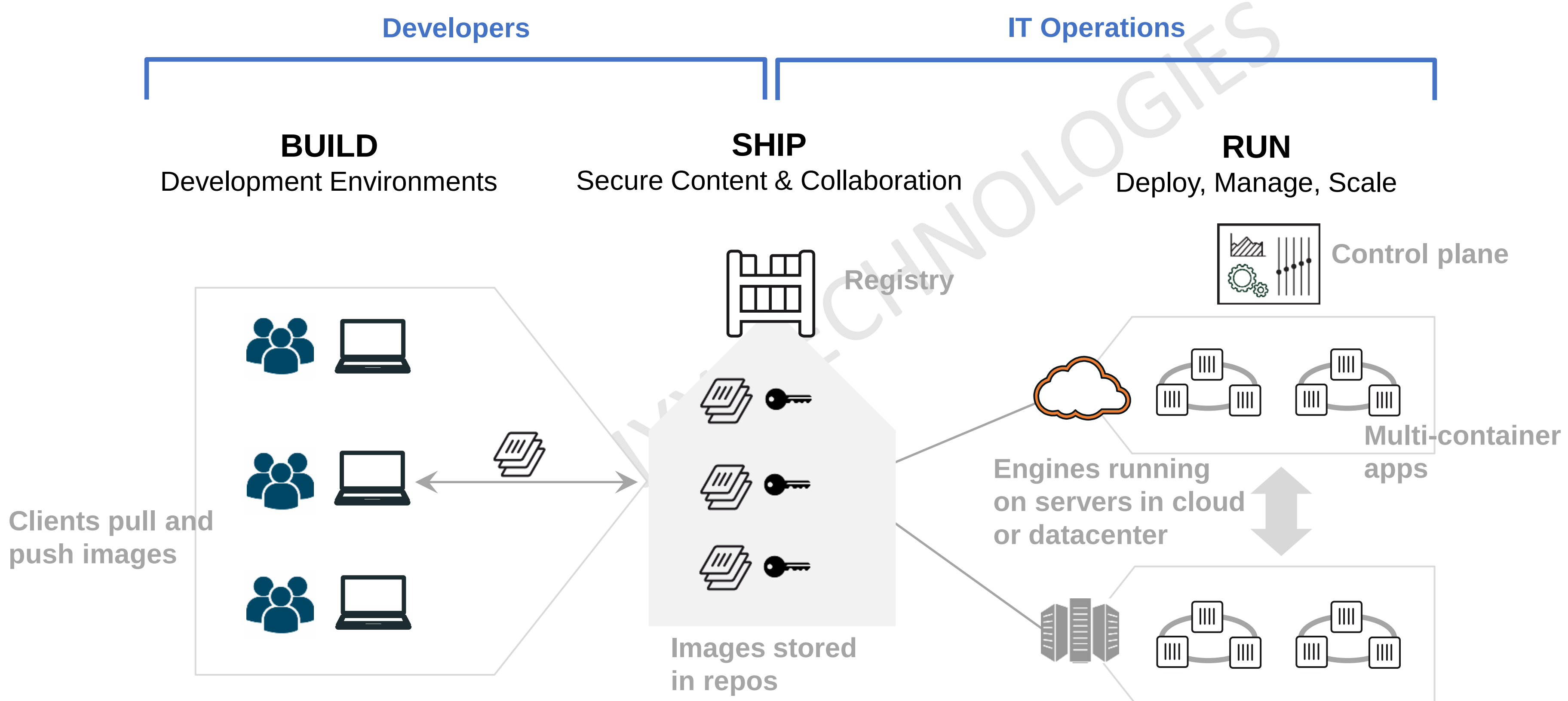
Docker Features

- Lightweight
    - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.

- Open
    - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.

- Secure
    - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

Container as a Service



Classroom Environment

Lab-1

- Create a ubuntu 20.04 server on your Azure account. Become root user by running sudo -i and perform rest of the steps.
- Install Docker software- apt-get install docker.io
- Check the status of Docker Services by running "systemctl status docker" (This should show you docker in active and running state)
- - Check the docker version using "docker --version"
- PS: Paste the output of docker --version on chat to me.

Docker Command Reference

- Below is the official Docker command reference link:
- https://docs.docker.com/engine/reference/commandline/container/#child-commands

Containers

Container Lifecycle

Format: docker <object> <subcommand> [options] <args>
-i -t -d

-i tells Docker to connect us to the container's stdin.
-t tells Docker that we want a pseudo-terminal.
-d run container in background and print container ID

1. Pull the image: docker image pull httpd:latest

2. Create a container: docker container run -itd httpd:latest

3. List the containers:
   docker container ls (this will show all the running containers)
   docker container ls –a (this will show all the stopped and running containers)

4. Stop the container: docker container stop <container_name>

5. Start the container: docker container start <container_name>

6. Login inside the container: docker container exec -it <container_name> /bin/bash
   exit

7. Create a container with a name: docker container run –itd –name=deepthicon httpd:latest

8. Rename a container: docker container rename <old_container_name> <new_container_name>

9. Remove the container : STOP and REMOVE : docker container stop <container_name> and docker container rm <container_name>

List Running Containers

- With docker ps, just like the UNIX ps command, lists running processes.

```
docker ps
docker ps -l
docker ps -a
```

- The (truncated) ID of our container.
- The image used to start the container.
- That our container has been running (Up) for a couple of minutes.
- Now, start multiple containers and use “docker ps” to list them.

Logs of Container

- Logs of container can be seen using:

```
docker logs 068 [[where 068 is prefix of ID]]
```

- We specified a *prefix of the full container ID.*
- You can, of course, specify the full ID.
- The logs command will output the *entire logs of the container.*
- To avoid being spammed with pages of output, we can use the --tail option:

```
docker logs --tail 3 068
```

Logs of Container

- We can also follow the logs real time using:

```
docker logs --tail 1 --follow 068
```

- This will display the last line in the log file.
- Then, it will continue to display the logs in real time.
- Use ^C to exit.

Stop our Container

- There are two ways we can terminate our detached container.

  - Killing it using the docker “kill” command.
  - Stopping it using the docker “stop” command.

- The first one stops the container immediately, by using the KILL signal.

- The second one is more graceful. It sends a TERM signal, and after 10 seconds, if the container has not stopped, it sends KILL.

LAB1

- -> Create your Docker Hub account (hub.docker.com)
- -> Login into your docker hub account

Note: Try using play with docker using your docker hub credentials and confirm that you are able to access the platform.

- -> Search an image with a name nginx from global repo
- -> Download the latest nginx image to your docker host
- -> List the image on your docker host
- -> Download another available version of nginx image (any other version)
- -> List the images again

- Show me the work -> docker image ls -->paste this out to me on chat

- -> Inspect your image for long image id
- -> Remove all local images from docker host
- -> List the images and verify the output

- **Useful commands -> docker image pull -> docker image ls -> docker image inspect → docker image rm**

LAB2

- --> Create a container named "dockercon" using nginx:latest
- Hint: docker container run -itd --name=dockercon nginx:latest

- --> Check the container status (docker container ls and docker container ls -a)

- Show me the work done --> Paste docker container ls output to me on chat

- --> Stop a container
- --> start a container
- --> login inside a container using "docker container exec -it <container-name> /bin/bash" ---> or you can use /bin/sh if bash wont work
- --> Try renaming a container from "dockercon" to “deepthicon"
- --> Try removing a running “deepthicon" container
- --> Finally stop a container and remove a container
- --> Remove all other containers also
- --> Remove all local available images

- **Useful commands -> docker container run, docker container ls, docker container stop, docker container start, docker container rm**

LAB3

- Create a new container named "webcon" using nginx image (application/container port is 80) and expose it on port number 83 of your server.
- docker container run -itd --name=webcon -p 83:80 nginx:latest
- Configure the network settings to allow port 83
- Go to browser put your public ip followed by port number 83
- Public-ip:83

Docker - Images

Store & manage images

- Images can be stored:
  - On your Docker host.
  - In a Docker registry.
- You can use the Docker client to download (pull) or upload (push) images.
- To be more accurate: you can use the Docker client to tell a Docker server to push and pull images to and from a registry.
- Lets explore docker public registry called “docker hub”

Downloading images

- There are two ways to download images.
  - Explicitly, with “docker pull”.
  - Implicitly, when executing “docker run” and the image is not found locally.
- Pulling an image.

```
docker pull debian:jessie
```

- Images can have tags.
- Tags define image versions or variants.
- “docker pull ubuntu” will refer to “ubuntu:latest”.
- The :latest tag is generally updated often.

LAB2

- Pull an image deepthianarayan/docker-image:latest from dockerhub
- Create a container named “mycon” in detached modes and expose the same at port no. 82 of your docker host.
- Hint: The container port is 80
- Test the container access from your browser
- Stop the container and remove the container
- Remove the image from docker host

Docker Image Lifecycle

Format : docker <object> <subcommand> [options] <args>

1. Search: hub.docker.com
2. Download an image: `docker image pull ubuntu:latest`
3. Download the specific version: `docker image pull ubuntu:18.04`
4. List the local images: `docker image ls`
5. Tag an image : `docker image tag ubuntu:latest deepthiimage:v1`
6. In docker, everything is an ID: Long-64chr and short-12chr
7. Delete an image: `docker image rm ubuntu:latest`

Docker Image Lifecycle

- The best part is though you have removed the image forcefully, but this will not impact the current container as the container preserves the metadata in containers folder.

- You containers are safe!.

Docker Image and Container Lifecycle - Extended

- Pull ubuntu image with 18.04 tag
- Create one Ubuntu container with name “testagain” using the same image
- Try to remove the image (you will get an error)
- Now stop your running container
- This time Docker will allow you to remove the image (SOMETIMES FORCEFULLY)
- NOW QUESTION IS WHAT WILL HAPPEN IF I WILL TRY TO START MY CONTAINER
- Start your container and container will be started
- Magic of Docker again but this time the “docker image ls will not show you anything”
- But one thing is sure that our running containers are Safe!

Building Images

Dockerfile overview

- A Dockerfile is a build recipe for a Docker image.
- It contains a series of instructions telling Docker how an image is constructed.
- The “docker build” command builds an image from a Dockerfile.

First Dockerfile

- Create a directory to hold our Dockerfile.

```
mkdir myimage
```

- Create a Dockerfile inside this directory.

```
cd myimage
vim Dockerfile
```

- Write below in our Dockerfile

```
FROM centos
RUN yum update -y
RUN yum -y install httpd
```

First Dockerfile

- “FROM” indicates the base image for our build.
- Each “RUN” line will be executed by Docker during the build.
- Our “RUN” commands **must be non-interactive.**
- No input can be provided to Docker during the build.

First Dockerfile

- Build the Dockerfile:

  ```
  docker build -t httpd .
  ```

- -t indicates the tag to apply to the image.

- . indicates the location of the Directory of Dockerfile.

Run & Tag the image

- Let’s run the new images:

  docker run -it <newImageId>

  rpm –qa | grep –i httpd

Real Example-1

- **Dockerfile:**

```
FROM centos:7
RUN yum -y update
RUN yum -y install httpd
RUN echo "<h1>Hello All, Hope you are enjoying learning Docker!</h1>" > /var/www/html/index.html
EXPOSE 80
CMD httpd -D FOREGROUND
```

- **Build the image:**

```
docker build -t deepthicent .
```

- **Testing**:

```
docker run -it -p 8888:80 deetphicent
```

Uploading images to Docker Hub

- We can share our images through the Docker Hub.

- Below are steps:
  - Crate an account on the Docker Hub
  - Login to docker hub- docker login
  - Tag our image accordingly (i.e. username/imagename): docker tag docker-image :latest deepthianarayan/docker-image:latest
  - docker push username/imagename: docker push deepthianarayan/docker-image:latest

- Anybody can now docker run deepthianarayan/docker-image:latest from any Docker host.

Container Network Model

Container Network Model

- The CNM was introduced in Engine 1.9.0 (November 2015).
- The CNM adds the notion of a network, and a new top-level command to manipulate and see those networks: “docker network”

```
docker network ls
```

What’s in a Network

- Conceptually, a network is a virtual switch.
- It can be local (to a single Engine) or global (across multiple hosts).
- A network has an IP subnet associated to it.
- A network is managed by a *driver.*
- A network can have a custom IPAM (IP allocator).
- Containers with explicit names are discoverable via DNS.

Creating a Network

- Let's create a network called dev.

```
docker network create dev
```

- The network can be seen with “network ls” command.

- We will create a named container on this network.

```
docker run -dt --name search --net dev tomcat
```

Lab

- Create two container (first one with name firstcon and second one with secondcon) using centos image
- Go inside firstcon and ping the secondcon (WITH NAME) i.e ping secondcon
- Make a note of the result in notepad I will come and check
- Repeat the same from secondcon and ping firstcon (WITH NAME) i.e ping firstcon
- Make a note of the result in notepad I will come and check

Lab

- docker run -itd --name=firstcon centos:latest
- docker run -itd --name=secondcon centos:latest
- docker container exec -it firstcon /bin/bash
  ping secondcon
- docker container exec -it secondcon /bin/bash
  ping firstcon

Networks in Docker

Creation of a network: docker network create deepthinet

List your networks: docker network ls

Details about your network: docker network inspect deepthinet

Create third container in deepthinet network: docker container run -itd --name=thirdcon --net=deepthinet centos:7

Create fourth container in deepthinet network: docker container run –itd --name=fourthcon --net=deepthinet centos:7

Test the communication: docker container exec -it thirdcon /bin/bash
ping fourthcon
exit

Repeat the test: docker container exec -it fourthcon /bin/bash
ping thirdcon
exit

Docker - Volumes

Working with Volumes

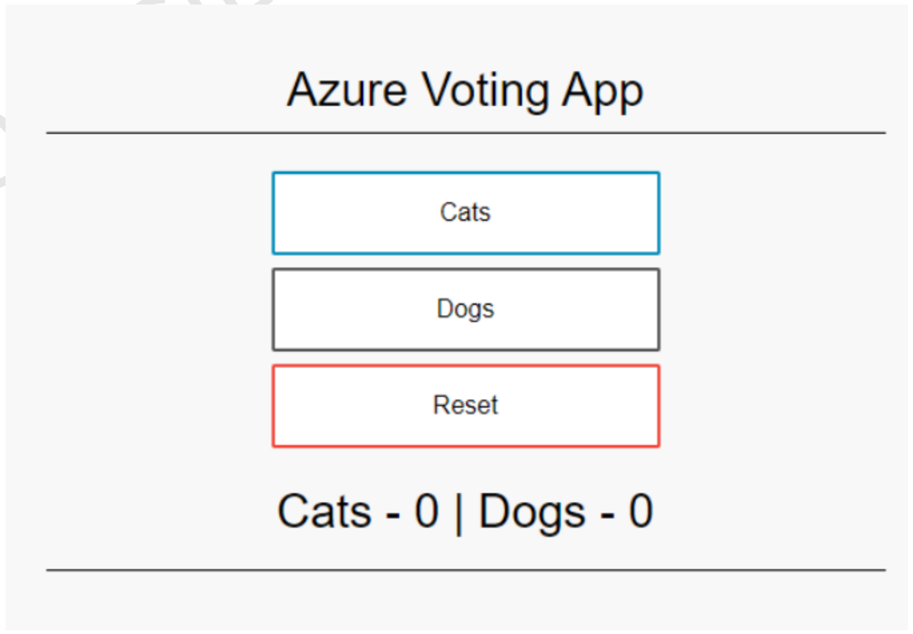
- Containers have ephemeral storage
- Data shared within the container is not accessible after it is terminated
- Volumes provide persistence to containers
- Each volume is implicitly or explicitly mapped to a host directory

Lab

- df -h /var/lib/docker
- docker volume create deepvol
- docker volume ls
- docker container run -itd --name=voltest -v deepvol:/deepapp centos:7
- docker container ls
- docker container exec -it voltest /bin/bash
- cd deepapp
- vi file1
- cat file1
- df -h
- exit
- docker container stop voltest
- docker container rm voltest
- Create another container using the same volume and test if the old file still exist

LAB - Voting App

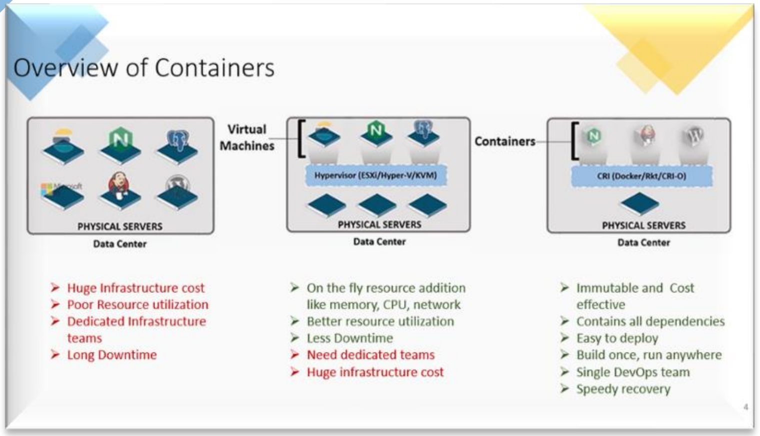
1. Fire up a VM
2. git clone https://github.com/Azure-Samples/azure-voting-app-redis.git
3. cd azure-voting-app-redis
4. docker-compose up -d
5. docker images
6. docker ps
7. docker-compose down



Bonus

- Docker images
- Actual size consumed by the OS for docker file:
- docker system df
- docker system df -v

Summary



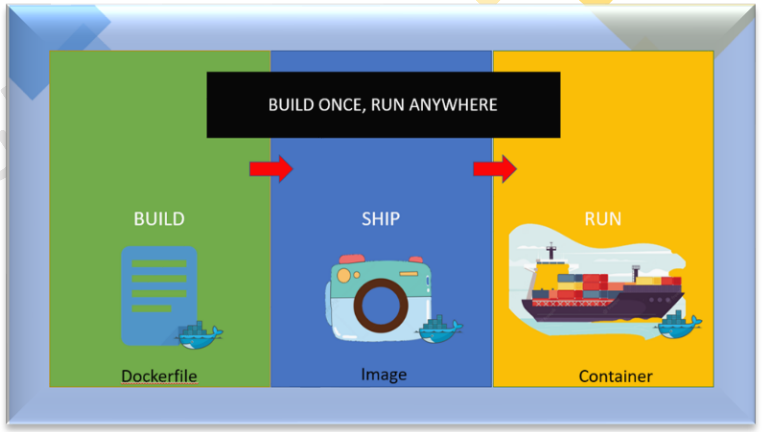
Overview of Containers

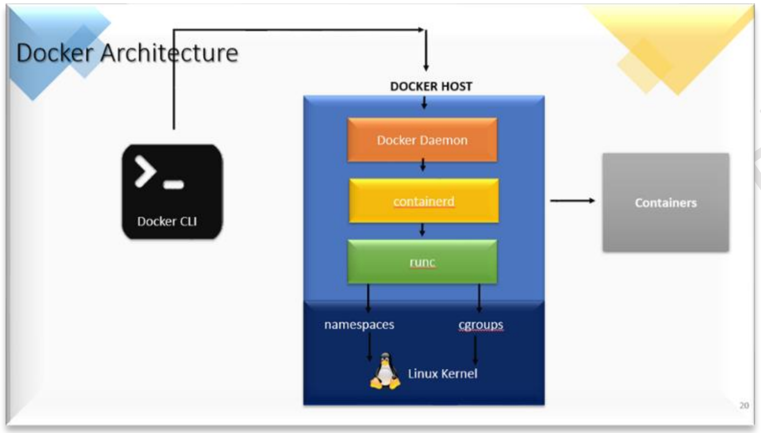
- Huge Infrastructure cost
- Poor Resource utilization
- Dedicated Infrastructure teams
- Long Downtime

- On the fly resource addition like memory, CPU, network
- Better resource utilization
- Less Downtime
- Need dedicated teams
- Huge infrastructure cost

- Immutable and Cost effective
- Contains all dependencies
- Easy to deploy
- Build once, run anywhere
- Single DevOps team
- Speedy recovery

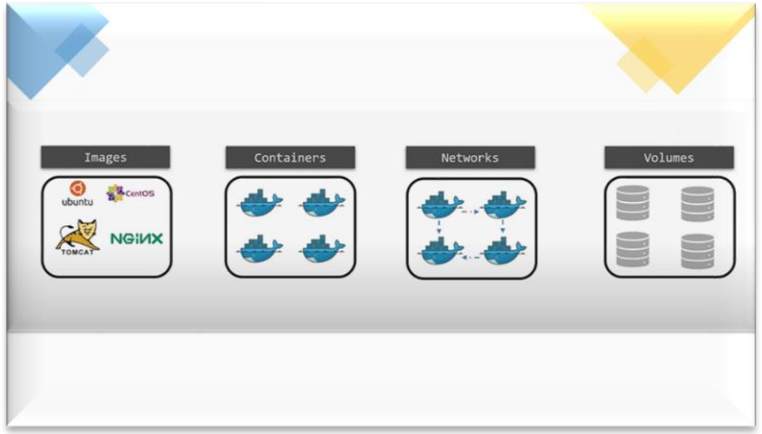
4





Docker Architecture

20



thank you!