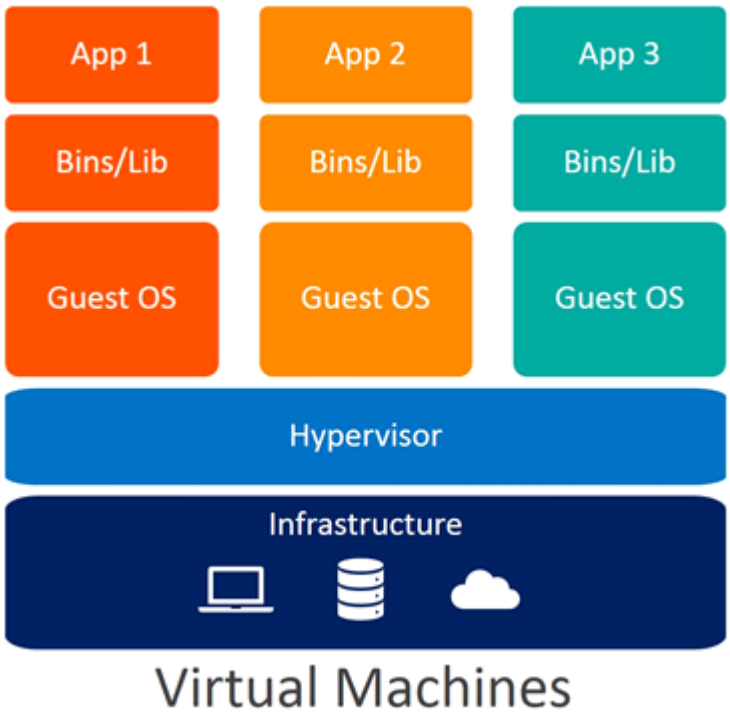
1. What is Virtualization?

Friday, May 19, 2023 4:44 PM

- Virtualization is the technique of importing a Guest operating system on top of a Host operating system.
- This technique was a revelation at the beginning because it allowed developers to run multiple operating systems in different virtual machines all running on the same host.
- This eliminates the need for extra hardware resource.

What is a Virtual Machine?

- Virtual machines are heavy software packages that provide complete emulation of hardware devices like CPU, Disk and Networking devices.
- Virtual machines may also include a complementary software stack to run on the emulated hardware.
- These hardware and software packages combined produce a fully functional snapshot of a computational system.



Virtual Machines

2. What is a container?

Friday, May 19, 2023 4:45 PM

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure.

- Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Containerization

- Containerization is the packaging together of software code with all it's necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own "container.“

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

3. What is Docker?

Friday, May 19, 2023 4:45 PM

- Docker is a software platform that allows you to build, test, and deploy applications quickly.
- Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.
- Build and run an image as a container.
- Share images using Docker Hub.

4. What is a Docker container?

Friday, May 19, 2023 4:46 PM

- A Docker container is a lightweight, standalone, and executable software package that includes everything needed to run an application, such as code, libraries, system tools, and settings.

- Containers are created using Docker, an open-source platform for building, shipping, and running applications in containers.

- Containers are isolated from the underlying operating system and other applications running on the same host, which helps to prevent conflicts between different applications and ensures that they have access to the resources they need without interfering with other applications.

- Docker containers are based on images, which are templates that define the contents of a container.

- Docker images can be created manually or automatically using Dockerfiles, which are text files that specify the steps needed to build an image. Once an image is created, it can be used to create one or more containers, each of which runs a separate instance of the application.

Docker containers that run on Docker Engine:

- **Standard**: Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight**: Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure**: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

Docker creates simple tooling and a universal packaging approach that bundles up all application dependencies inside a container which is then run-on Docker Engine.

Docker Engine enables containerized applications to run anywhere consistently on any infrastructure, solving "dependency issues" for developers and operations teams, and eliminating the "it works on my laptop!" problem.

5. Virtual Machine vs Docker container

Friday, May 19, 2023 4:46 PM

Virtual machines (VMs) and Docker containers are both technologies used for running applications in an isolated environment, but they have some significant differences.

A virtual machine:

- is an emulation of a complete physical machine, including a full operating system, hardware resources, and an application.
- Each VM runs its own copy of the operating system and is isolated from the host machine and other VMs.
- This isolation makes VMs very secure, but also resource-intensive, as each VM requires its own copy of the operating system and resources, such as CPU, memory, and disk space.

A Docker, on the other hand:

- is a containerization technology
- that provides a way to run applications in isolated environments without the overhead of a full operating system.
- Docker containers share the same kernel as the host machine, which makes them much more lightweight and efficient than VMs. Docker containers are also more portable than VMs, as they can be easily moved from one environment to another.

Here are some key differences between VMs and Docker containers:

Resource utilization:

- VMs require more resources than Docker containers, as they run a complete operating system in each instance.
- Docker containers, on the other hand, share the same kernel as the host machine and only require resources for the application and its dependencies.

Isolation:

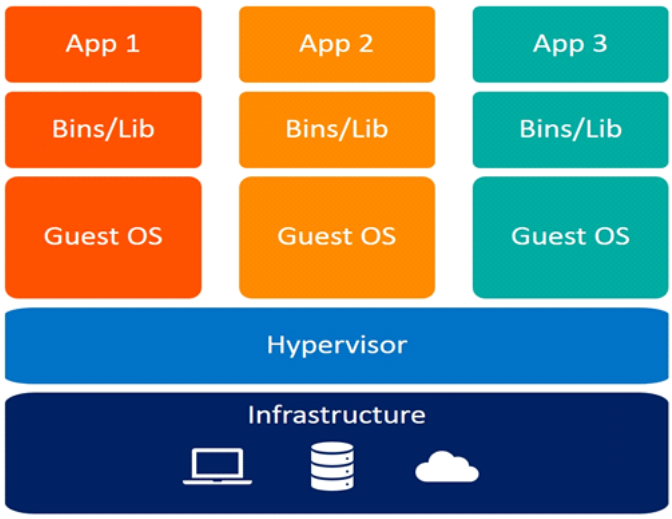
- VMs provide complete isolation from the host machine and other VMs, which makes them more secure, but also less efficient.
- Docker containers share the same kernel as the host machine and are more lightweight and efficient, but may not provide the same level of isolation as VMs.

Portability:

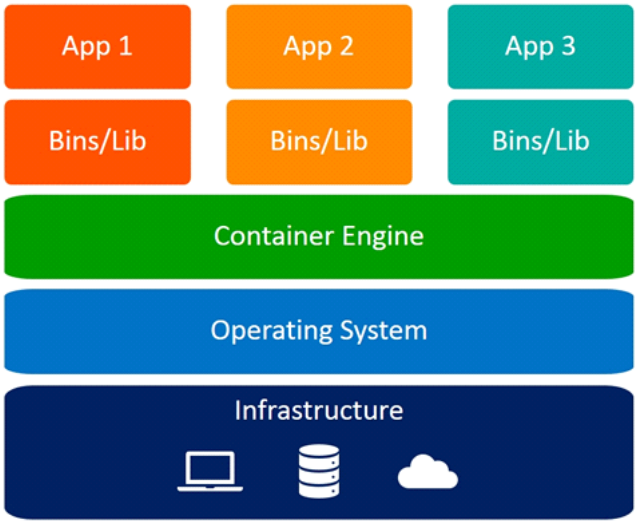
- Docker containers are more portable than VMs, as they can be easily moved from one environment to another.
- VMs, on the other hand, are less portable, as they may require different hardware or configurations in different environments.

Key	**Virtual Machine**	**Docker**
Resource utilization	VMs require more resources than Docker containers, as they run a complete operating system in each instance.	Docker containers, on the other hand, share the same kernel as the host machine and only require resources for the application and its dependencies.
Isolation	VMs provide complete isolation from the host machine and other	Docker containers share the same kernel as the host machine and are more

	VMs, which makes them more secure, but also less efficient.	lightweight and efficient, but may not provide the same level of isolation as VMs.
Portability	Docker containers are more portable than VMs, as they can be easily moved from one environment to another.	VMs, on the other hand, are less portable, as they may require different hardware or configurations in different environments.



Virtual Machines



Containers

6. When to use Containers?

Friday, May 19, 2023 4:46 PM

Containers are a good choice for the majority of application workloads. Consider containers in particular if the following is a priority:

Start time	• Docker containers typically start in a few seconds or less, whereas virtual machines can take minutes.
Efficiency	• Because Docker containers share many of their resources with the host system, they require fewer things to be installed in order to run. • A container typically takes up less space and consumes less RAM and CPU time. Due to this, you can fit more applications on a single server using containers. • containers may help to save money on cloud computing costs.
Licensing	• Most of the core technologies required to deploy Docker containers & Kubernetes, are free and open source, which is cost effective.
Code reuse	• Each running container is based on a container image, which contains the binaries and libraries that the container requires to run a given application. Container images are easy to build using Dockerfiles. • They can be shared and reused using container registries, which are basically repositories that host container images. • You can set up an internal registry to share and reuse containers within your company. • Thousands of prebuilt images can be downloaded from public registries (e.g. Docker Hub or Quay.io) for free and used as the basis for building your own containerized applications.

7. When to stick with virtual machines?

Friday, May 19, 2023 4:47 PM

Security	• Virtual machines are more isolated from each other and from the host system than are Docker containers. • Virtual machines are arguably more secure overall than containers.
Linux and Windows portability	• Docker is not as portable. Although in some ways Docker reduces dependence on your operating system. • Docker containers for Linux only work on Linux hosts, and the same holds true for Windows.
Rollback features	• Virtual machine platforms make it easy to “snapshot” virtual machines at a given point in time, and to “roll back” a machine when desired. • Docker doesn’t offer the same type of functionality. • You can roll back container images, but because containers store their data outside of the image in most cases, rolling back an image won’t help you recover data that was lost by a running application.

8. Why Switch to Docker?

Friday, May 19, 2023 4:47 PM

It works on my machine!!!

- This is common statement, given by the developer to another.
- This happens when you transfer or share the application you built to another person & he/she tries running the same application on their machines.
- The reason why this is happening are like:
  - One or more files missing.
  - Software version mismatch – MAJORILY OCCURING.
  - Different configuration settings.
- Solution is DOCKER.

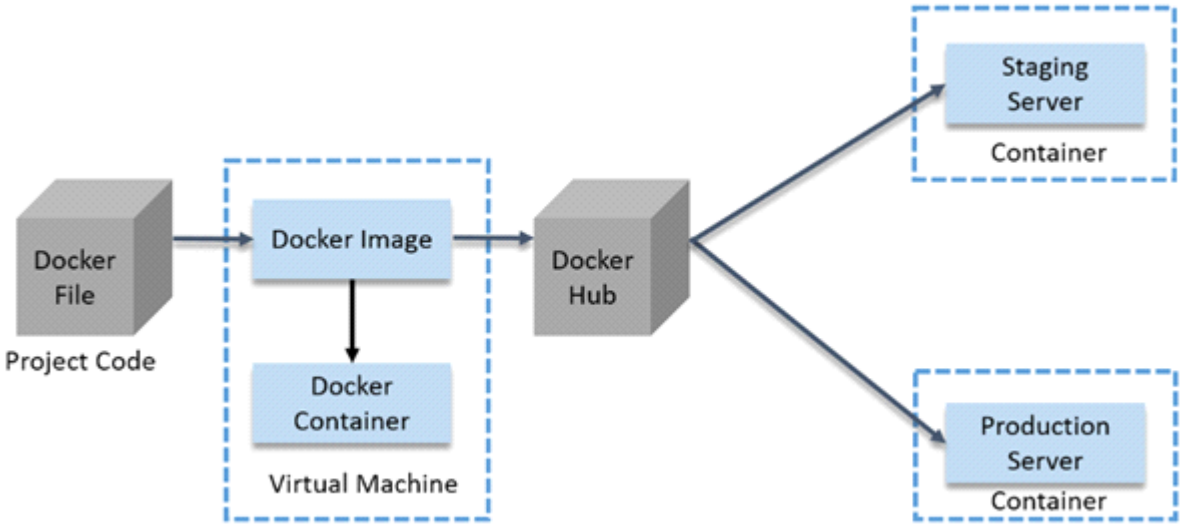
9. Terminologies of Docker

Friday, May 19, 2023 4:47 PM

1. **A Docker Image** - is created by the sequence of commands written in a file called as Dockerfile. When this Image is executed by “docker run” command it will by itself start whatever application or service it must start on its execution.
2. **Dockerfile** - When this Dockerfile is executed using a docker command it results into a Docker Image with a name.
3. **Docker Containers** - are a lightweight solution to Virtual Machines
4. **Docker Hub**
    - is like GitHub for Docker Images.
    - It is basically a cloud registry where you can find Docker Images uploaded by different communities or even by yourself.
5. **Docker Compose**
    - Docker Compose is basically used to run multiple Docker Containers as a single server.
6. **Docker Client**
    - The command line tool that allows the user to interact with the daemon.
7. **Docker Daemon**
    - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to.

10. How a Docker Container Works?

Friday, May 19, 2023 4:55 PM



1. A developer will first write the project code in a Docker file and then build an image from that file.
2. This image will contain the entire project code.
3. Now, you can run this Docker Image to create as many containers as you want.
4. This Docker Image can be uploaded on Docker hub (It is basically a cloud repository for your Docker Images, you can keep it public or private).
5. This Docker Image on the Docker hub, can be pulled by other teams such as QA or Prod.

Detailed work flow of a docker container?

1. **Dockerfile**: The process starts with creating a Dockerfile, which is a text file that contains instructions for building a Docker image. The Dockerfile specifies the base image, any additional dependencies, environment variables, and commands to run when the image is built.

2. **Docker Build:** Using the Docker CLI, you initiate the build process by running the docker build command and specifying the path to the directory containing the Dockerfile. Docker then reads the Dockerfile and builds an image according to the specified instructions.

3. **Image Layers**: During the build process, Docker uses a layered file system to optimize resource usage and enable efficient image sharing. Each instruction in the Dockerfile creates a new layer in the image. Layers are cached and can be reused in subsequent builds, speeding up the process.

4. **Docker Registry**: Once the image is built, it can be optionally pushed to a Docker registry. A registry is a centralized repository for storing and sharing Docker images. Common registries include Docker Hub, which is a public registry, and private registries that can be hosted locally or on cloud platforms.

5. **Docker Run**: To create and start a container from an image, you use the docker run command. This command specifies the image to use and any additional runtime configuration, such as port mappings, volume mounts, environment variables, and networking settings.

6. **Container Execution**: When the container is started, Docker creates an isolated environment based on the image's specifications. It sets up a separate file system, network namespace, and process space

for the container. The container runs a specific process or command defined in the image, and it can have its own set of resources allocated, such as CPU and memory.

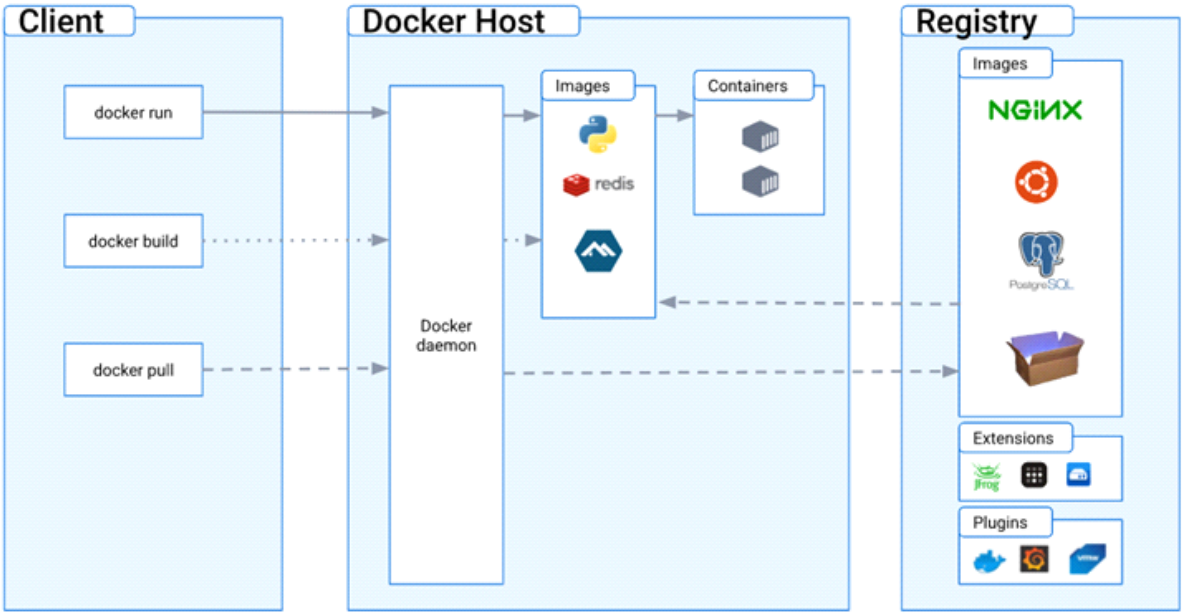
7. **Container Interaction**: Once the container is running, you can interact with it in various ways. You can access the container's console or shell, view its logs, copy files into or out of the container, and execute commands within the container using the Docker CLI.

8. **Container Lifecycle**: Containers can be paused, stopped, restarted, or removed using Docker commands. Pausing a container suspends its processes, while stopping it terminates them. Restarting a container restarts its processes. Removing a container deletes it and frees up the resources it was using.

9. **Container Orchestration**: In more complex scenarios, where multiple containers need to work together as part of a distributed application, container orchestration platforms like Docker Swarm or Kubernetes are used. These platforms provide advanced features such as service discovery, load balancing, scaling, and automated deployment of containers.

11. Docker architecture & components

Friday, May 19, 2023 5:14 PM



Docker architecture is composed of several components that work together to provide a platform for building, shipping, and running applications in containers. Here are the main components of the Docker architecture:

- Docker Engine
  - The Docker Engine is the core component of the Docker platform.
  - It's responsible for managing the lifecycle of containers, including starting, stopping, and restarting them.
  - The Docker Engine also provides an API and a command-line interface (CLI) for interacting with Docker.
- Docker images
  - Docker images are templates that define the contents of a container, including the application code, libraries, and system tools.
  - Docker images are created using a Dockerfile, which is a text file that specifies the steps needed to build the image.
- Docker containers
  - Docker containers are instances of Docker images that are running in an isolated environment.
  - Each container runs a separate instance of the application and has its own file system, network interface, and resource allocation.
- Docker registry
  - A Docker registry is a central repository where Docker images can be stored and shared between users.
  - Docker Hub is the default registry for Docker, but there are other options available, such as private registries that can be hosted on-premises or in the cloud.
- Docker CLI
  - The Docker CLI is a command-line interface that allows users to interact with the Docker Engine and perform common tasks, such as building images, running containers, and managing networks.
- Docker API

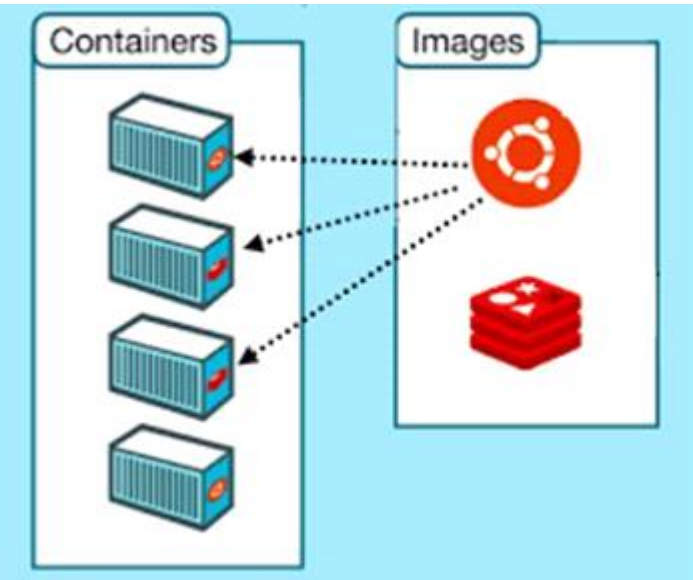
- The Docker API provides a programmatic interface for interacting with Docker. Applications can use the Docker API to automate the creation, deployment, and management of Docker containers.

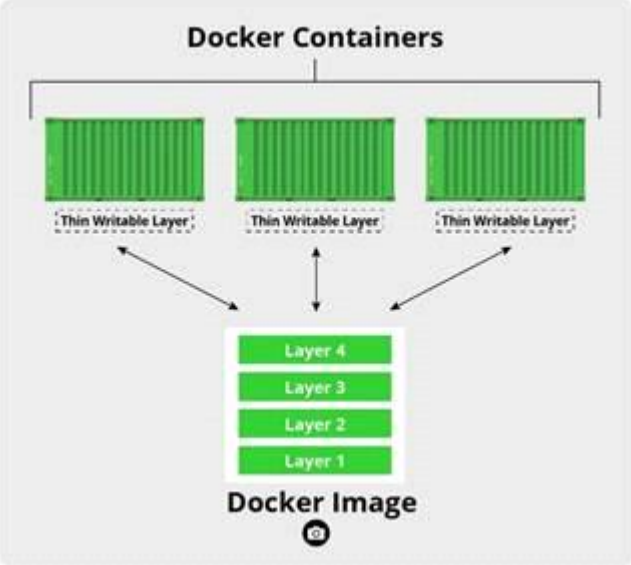
Explanation of each component in the diagram:

- The Docker daemon
  - The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
  - A daemon can also communicate with other daemons to manage Docker services.
- The Docker client
  - The Docker client (docker) is the primary way that many Docker users interact with Docker.
  - The client sends these commands to dockerd, which carries them out.
  - The docker command uses the Docker API.
  - The Docker client can communicate with more than one daemon.
- Docker Desktop (when your system/machine is old)
  - Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices.
  - Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper.
- Docker registries
  - A Docker registry stores Docker images.
  - Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  - You can even run your own private registry.
  - When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.
- Docker objects
  - When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

12. Images & Containers:

Friday, May 19, 2023 5:22 PM





Containers: contains application, environment.

Images: the templates/blueprint for container, an image contains code & app. Container runs this code. This image can create multiple times.

13. Docker Networking

Friday, May 19, 2023 5:23 PM

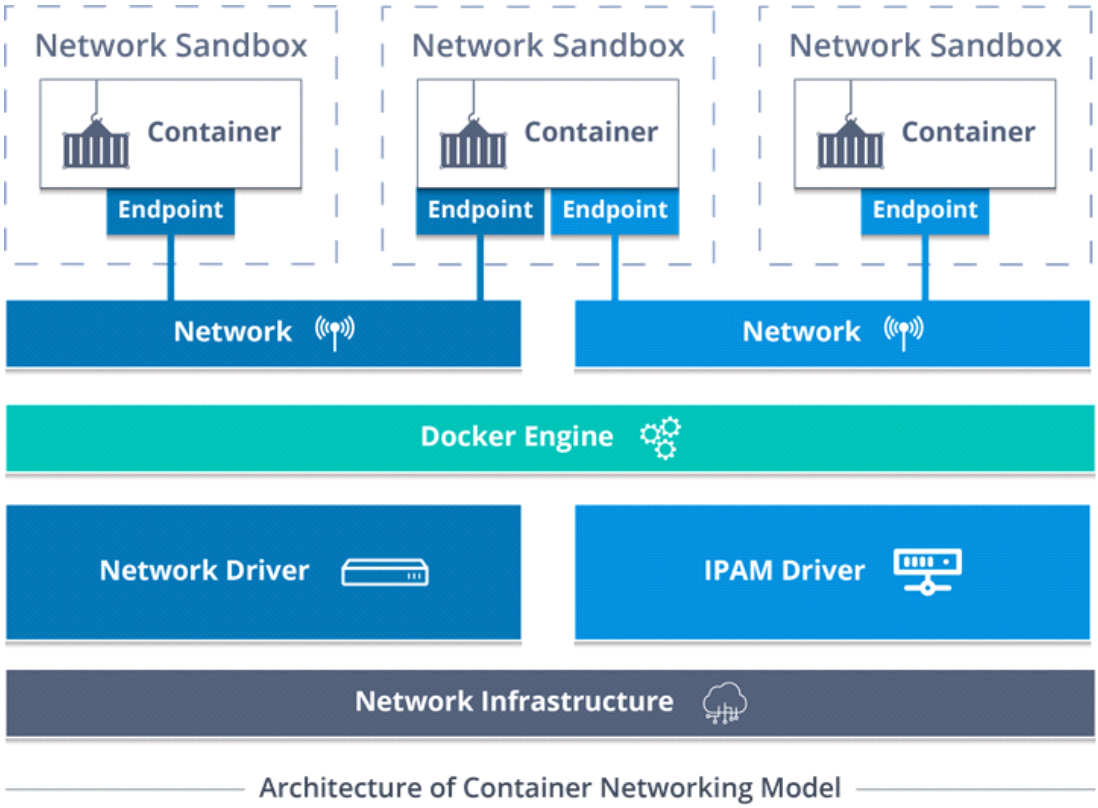
Docker Networking refers to the networking capabilities and features provided by Docker, an open-source containerization platform. Docker Networking allows containers to communicate with each other and with the outside world, enabling seamless connectivity and network integration for applications running in Docker containers.

Here are some key concepts and features related to Docker Networking:

- ✓ **Containers**:
  - Docker allows you to create and run containers, which are lightweight, isolated environments that package an application and its dependencies.
  - Each container can have its own network stack, including network interfaces, IP addresses, and routing tables.
- ✓ **Docker Network**:
  - A Docker network is a virtual network that provides communication between containers.
  - Docker offers various network drivers, such as bridge, overlay, host, and macvlan, each with its own characteristics and use cases.
  - By default, Docker creates a bridge network called "bridge" that allows containers to communicate with each other.
- ✓ **Bridge Network**:
  - The bridge network driver connects containers to a bridge on the host.
  - Containers attached to the same bridge can communicate with each other using IP addresses.
  - This is the default network driver for Docker containers and is suitable for most use cases.
- ✓ **Overlay Network**:
  - The overlay network driver allows containers to communicate across multiple Docker hosts.
  - It helps in the creation of distributed applications spanning multiple machines or even different data centers.
  - It uses a network overlay technology, such as VXLAN, to encapsulate and transmit network traffic between hosts.
- ✓ **Container-to-Container Communication**:
  - Containers within the same Docker network can communicate with each other using their IP addresses or container names.
  - Docker assigns each container a unique IP address within the network, allowing them to discover and communicate with other containers by their IP or hostname.
- ✓ **Exposing Ports**:
  - Docker allows you to expose ports on containers to the host system or external networks.
  - This enables access to containerized applications or services from the outside world. You can specify port mappings when running a container to bind a container port to a host port.
- ✓ **DNS Resolution**:
  - Docker provides an embedded DNS server that allows containers to resolve each other's IP addresses using container names.
  - This enables seamless service discovery and communication between containers without the need to know the IP addresses explicitly.

✓ **Docker Compose**:

- Docker Compose is a tool that simplifies the orchestration of multi-container applications.
- It provides a way to define and manage multiple containers as a single service using a YAML file.
- Docker Compose allows you to specify networking configurations, including network creation, container-to-container communication, and port mappings.



Architecture of Container Networking Model

✓ **Network sandbox**

- It is isolated sandbox that holds the network configurations of containers
- Sandbox is created when a user requests to generate an endpoint on the network.

✓ **Endpoints**

- It can have several endpoints in a network.
- EP established connectivity for the container services with other services.

✓ **Network**

- Provides networking components

✓ **Docker Engine**

- Is the base engine installed on host machine to build & run containers using docker.

✓ **Network driver**

- Its task is to manage the network with multiple drivers.

✓ **IPAM drivers**

- Manages the allocation and assignment of IP addresses to containers within a Docker network.

✓ **Network architecture**

- Provides connectivity between endpoints

IPAM driver:

An IPAM (IP Address Management) driver is a component that manages the allocation and assignment of IP addresses to containers within a Docker network.

The IPAM driver is responsible for the following tasks:

1. **IP Address Allocation**: The IPAM driver manages the pool of available IP addresses within a Docker network. When a container is created, the IPAM driver assigns an IP address to it from the available pool. This ensures that containers are assigned unique and non-conflicting IP addresses.
2. **Subnet Management**: The IPAM driver defines the subnet configuration for the Docker network, including the range of IP addresses that can be assigned to containers. It ensures that the IP addresses allocated to containers fall within the defined subnet.
3. **Gateway Configuration**: The IPAM driver sets up the default gateway for the Docker network. The gateway is the IP address used by containers to reach external networks. The IPAM driver assigns the appropriate gateway IP address to containers when they are connected to the network.
4. **Custom IP Address Assignment**: Some IPAM drivers allow for custom IP address assignment, where you can specify a particular IP address to be assigned to a container. This can be useful when you require specific IP address management for certain containers.
5. **Integration with External IPAM Systems**: Docker IPAM drivers can integrate with external IP Address Management systems, such as IPAM systems used in large-scale network deployments. This allows Docker to leverage existing IPAM infrastructure and management processes.

14. Installing Docker & other packages:

Friday, May 19, 2023 5:30 PM

1. Install docker commands: (run as root user)

---

yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

- docker-ce
- docker-ce-cli
- containerd.io
- docker-buildx-plugin
- docker-compose-plugin

yum repolist
yum install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
which docker

2. verify if docker is running, else start it.

---

systemctl status docker
systemctl start docker
systemctl enable docker
systemctl status docker

3. verify docker image & container status

---

docker info

4. checking images

[root@docmaster ~]# #list images
[root@docmaster ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
[root@docmaster ~]# docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE

5. downloading the image(s)

---

docker pull ubuntu //this downloads latest ubuntu version
docker pull ubuntu:18.04 //this downloads specific ubuntu version

15. Docker basic commands

Friday, May 19, 2023 5:39 PM

Here are some basic Docker commands that you can use to work with Docker containers:

docker run: Creates and runs a new container based on an image.
Example: - Runs an interactive Ubuntu container and opens a bash shell.
docker run -it ubuntu:latest /bin/bash

docker ps: Lists running containers.
Example: - Lists all running containers.
docker ps

docker images: Lists available Docker images.
Example: - Lists all locally available Docker images.
docker images

docker pull: Downloads an image from a Docker registry.
Example: - Downloads the latest Nginx image from Docker Hub.
docker pull nginx:latest

docker build: Builds an image from a Dockerfile.
Example: - Builds an image named "myimage" using the Dockerfile in the current directory.
docker build -t myimage:latest .

docker stop: Stops a running container.
Example: - Stops the container with the specified ID.
docker stop container_id

docker start: Starts a stopped container.
Example: - Starts the container with the specified ID.
docker start container_id

docker restart: Restarts a running container.
Example: - Restarts the container with the specified ID.
docker restart container_id

docker rm: Removes a container.
Example: - Removes the container with the specified ID.
docker rm container_id

docker rmi: Removes an image.
Example: - Removes the image with the specified ID.
docker rmi image_id

docker exec: Executes a command inside a running container.
Example: - Executes an interactive bash shell inside the running container.
docker exec -it container_id /bin/bash

docker logs: Displays the logs of a container.
Example: - Shows the logs of the container with the specified ID.
docker logs container_id

16. Docker basic commands (with snapshots)

Friday, May 19, 2023 5:51 PM

I have already installed DOCKER CE on Ubuntu 20.04 on Azure VM.

To list docker images details (like: running, stopped, paused & downloaded images).
#docker info

```
root@dockerdemo:~# docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.9.1-docker)
  compose: Docker Compose (Docker Inc., v2.12.2)
  scan: Docker Scan (Docker Inc., v0.21.0)

Server:
 Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
 Images: 1
```

To list current images within docker:
docker image ls

```
root@dockerdemo:~# docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
ubuntu       14.04     13b66b487594   19 months ago    196MB
```

To download latest docker images from DockerHub:
docker pull alpine

```
root@dockerdemo:~# docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
213ec9aee27d: Pull complete
Digest: sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Thinking-Why Alpine: *alpine is a minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size.

To verify if the image is downloaded:
docker image ls

```
root@dockerdemo:~# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
alpine        latest    9c6f07244728   3 months ago     5.54MB
ubuntu        14.04     13b66b487594   19 months ago    196MB
root@dockerdemo:~#
```

To run the image as a container:
docker run -i -t alpine /bin/sh

```
root@dockerdemo:~# docker run -i -t alpine /bin/sh
/ # whoami
root
/ # hostname
efe08ac3ec31
/ # pwd
/
/ #
```

i = interaction
t = terminal access
alpine = image we downloaded
/bin/sh = default shell to write linux commands

To list docker images:
docker ps -a

```
root@dockerdemo:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                       PORTS     NAMES
efe08ac3ec31   alpine         "/bin/sh"     2 minutes ago    Exited (0) 18 seconds ago              zealous_solomon
2b4b92f119f2   alpine         "/bin/bash"   3 minutes ago    Created                                objective_edison
cb72731032c9   alpine         "/bin/bash"   3 minutes ago    Created                                relaxed_elbakyan
2ba5f78f8f61   ubuntu:14.04   "/bin/bash"   17 hours ago     Exited (127) 17 hours ago              unruffled_williamson
root@dockerdemo:~#
```

To run another images:
docker run -it ubuntu /bin/bash

```
root@dockerdemo:~# docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
e96e057aae67: Pull complete
Digest: sha256:4b1d0c4a2d2aaf63b37111f34eb9fa89fa1bf53dd6e4ca954d47caebca4005c2
Status: Downloaded newer image for ubuntu:latest
root@a5bd4d4b33ff:/#
```

Checking running containers:
docker ps -a

```
root@dockerdemo:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                       PORTS     NAMES
a5bd4d4b33ff   ubuntu         "/bin/bash"   10 minutes ago   Exited (127) 14 seconds ago            youthful_meitner
efe08ac3ec31   alpine         "/bin/sh"     2 hours ago      Exited (0) 2 hours ago                 zealous_solomon
2b4b92f119f2   alpine         "/bin/bash"   2 hours ago      Created                                objective_edison
cb72731032c9   alpine         "/bin/bash"   2 hours ago      Created                                relaxed_elbakyan
2ba5f78f8f61   ubuntu:14.04   "/bin/bash"   19 hours ago     Exited (127) 19 hours ago              unruffled_williamson
```

Starting & stopping docker container:
Starting →# docker start <container_id>
Stop →# docker stop <container_id>

```
root@dockerdemo:~# docker start a5bd4d4b33ff
a5bd4d4b33ff
root@dockerdemo:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS
a5bd4d4b33ff   ubuntu         "/bin/bash"   11 minutes ago   Up 4 seconds
efe08ac3ec31   alpine         "/bin/sh"     2 hours ago      Exited (0) 2 hours ago
2b4b92f119f2   alpine         "/bin/bash"   2 hours ago      Created
cb72731032c9   alpine         "/bin/bash"   2 hours ago      Created
2ba5f78f8f61   ubuntu:14.04   "/bin/bash"   19 hours ago     Exited (127) 19 hours ago
root@dockerdemo:~# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS              PORTS     NAM
a5bd4d4b33ff   ubuntu    "/bin/bash"   12 minutes ago   Up About a minute             you
root@dockerdemo:~# docker stop a5bd4d4b33ff
a5bd4d4b33ff
root@dockerdemo:~#
```

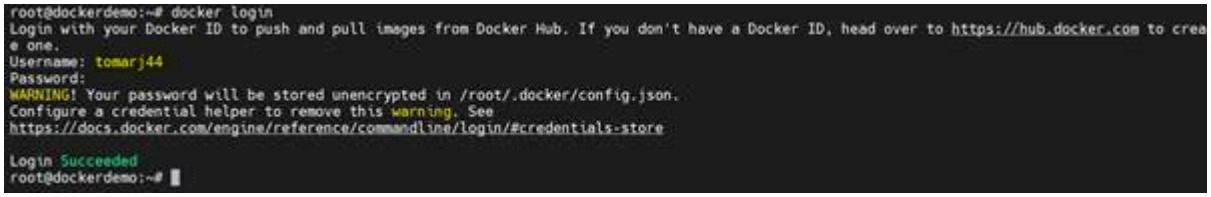
Get the docker stats while its running:
docker stats

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O   PIDS
a5bd4d4b33ff   youthful_meitner  0.00%     844KiB / 7.765GiB   0.01%     876B / 0B   0B / 0B     1
```

17. login to Docker hub

Friday, May 19, 2023 5:51 PM

To create a custom image, you need to login to docker (NOT MANDATORY):
docker login



18. Creating 1st Dockerfile (basic)

Friday, May 19, 2023 5:52 PM

Creating Dockerfile from scratch:

#vim Dockerfile

```
root@dockerdemo:~# cat Dockerfile
FROM ubuntu:16.04
COPY . /var/www/html
root@dockerdemo:~#
```

Building docker image using Dockerfile:

docker build -t dockerdemo-website .

```
root@dockerdemo:~# docker build -t dockerdemo-website .
Sending build context to Docker daemon  25.6kB
Step 1/2 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
58690f9b18fc: Pull complete
b51569e7c507: Pull complete
da8ef40b9eca: Pull complete
fb15d46c38dc: Pull complete
Digest: sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6
Status: Downloaded newer image for ubuntu:16.04
 ---> b6f507652425
Step 2/2 : COPY . /var/www/html
 ---> 03e0c3950141
Successfully built 03e0c3950141
Successfully tagged dockerdemo-website:latest
```

Listing created image:

docker image ls

```
root@dockerdemo:~# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
dockerdemo-website  latest    03e0c3950141   2 minutes ago    135MB
ubuntu              latest    a8780b506fa4   6 days ago       77.8MB
alpine              latest    9c6f07244728   3 months ago     5.54MB
ubuntu              16.04     b6f507652425   14 months ago    135MB
ubuntu              14.04     13b66b487594   19 months ago    196MB
root@dockerdemo:~#
```

Running the container, you just build:

docker run -it -d -p 80:80 dockerdemo-website

```
root@dockerdemo:~# docker run -it -d -p 80:80 dockerdemo-website
44447626cd93aa5462fab834207039f52108d18c2ca1bf95c8445325362c1405
```

Verifying the running container:

docker ps -a

```
root@dockerdemo:~# docker ps -a
CONTAINER ID   IMAGE              COMMAND       CREATED             STATUS                      PORTS
44447626cd93   dockerdemo-website "/bin/bash"   4 minutes ago       Up 4 minutes                0.0.0.0:80->80/tcp, :::80->80/tcp
a5bd4d4b33ff   ubuntu             "/bin/bash"   About an hour ago   Up 41 minutes
efe08ac3ec31   alpine             "/bin/sh"     3 hours ago         Exited (0) 3 hours ago
2b4b92f119f2   alpine             "/bin/bash"   3 hours ago         Created
cb72731032c9   alpine             "/bin/bash"   3 hours ago         Created
2ba5f78f8f61   ubuntu:14.04       "/bin/bash"   20 hours ago        Exited (127) 20 hours ago
root@dockerdemo:~#
```

19. Creating a new image using existing Ubuntu docker image

Friday, May 19, 2023 5:56 PM

Step1 – download/pull a new docker image from docker hub

docker pull ubuntu(already there for me)

```
root@dockerdemo:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:4b1d0c4a2d2aaf63b37111f34eb9fa89fa1b
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
```

Step2 – Run a new container (name - testUbuntu) with the downloaded ubuntu image & update it too

docker run --name testUbuntu -it ubuntu

```
root@dockerdemo:~# docker run --name testUbuntu  -it ubuntu
root@539abfb18af3:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [99.8 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [13.6 kB]
```

Step3 – verify if the container is present.

docker image ls

```
root@dockerdemo:~# docker image ls
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
dockerdemo-website   latest    03e0c3950141   6 hours ago      135MB
ubuntu               latest    a8780b506fa4   6 days ago       77.8MB
alpine               latest    9c6f07244728   3 months ago     5.54MB
hello-world          latest    feb5d9fea6a5   13 months ago    13.3kB
ubuntu               16.04     b6f507652425   14 months ago    135MB
ubuntu               14.04     13b66b487594   19 months ago    196MB
root@dockerdemo:~#
```

Step4 – Verify if GIT is installed or not

git –version

```
root@8431ab907f7f:/# git --version
bash: git: command not found
```

Step5 – Install GIT

apt-get install git

```
root@8431ab907f7f:/# apt-get install git
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
The following additional packages will be installed:
```

Step6 – Verify GIT version

git --version

```
root@8431ab907f7f:/# git --version
git version 2.34.1
```

Step7 – Exit the console

exit

Step8 – verify if the container is still running:

docker ps -a

```
root@dockerdemo:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NA
MES
8431ab907f7f   ubuntu    "bash"    3 minutes ago    Exited (0) 7 seconds ago              my
Ubuntu
```

Step9 – check all current images:

docker images

```
root@dockerdemo:~# docker images
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
dockerdemo-website   latest    03e0c3950141   6 hours ago      135MB
ubuntu               latest    a8780b506fa4   6 days ago       77.8MB
alpine               latest    9c6f07244728   3 months ago     5.54MB
hello-world          latest    feb5d9fea6a5   13 months ago    13.3kB
ubuntu               16.04     b6f507652425   14 months ago    135MB
ubuntu               14.04     13b66b487594   19 months ago    196MB
```

Step10 - create your own new image using this container & verify

docker commit myUbuntu custom-ubuntu

docker images

```
root@dockerdemo:~# docker commit myUbuntu custom-ubuntu
sha256:292c55511acf46f6e275afa3bc4e444ea62316161a0fa1aef184a9d0edb239a8
root@dockerdemo:~# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
custom-ubuntu       latest    292c55511acf   6 seconds ago    192MB
```

Step11 – Run another container using this same (custom-ubuntu) image

docker run -it --name test1 custom-ubuntu

```
root@dockerdemo:~# docker run -it --name  test1 custom-ubuntu
```

Step12 – verify GIT is present or not.

```
root@424c1b116098:/# git --version
git version 2.34.1
```

20. Creating a new image using Dockerfile

Friday, May 19, 2023 5:59 PM

Step1 – Create a "Dockerfile" using any editor.

vim Dockerfile

D in Dockerfile must be CAPITAL

```
FROM centos:7
MAINTAINER Jitendra jitendra@pioneerbusineessolutions.in
RUN yum makecache
RUN yum upgrade -y
RUN yum install -y httpd
RUN yum install -y git
RUN yum install -y vi
```

Step2 – build the docker image using the Dockerfile.

docker build -t test_centos:Docker .

```
root@dockerdemo:~# docker build -t test_centos:Dockerfile .
Sending build context to Docker daemon  30.72kB
Step 1/7 : FROM centos:7
7: Pulling from library/centos
2d473b07cdd5: Pull complete
Digest: sha256:c73f515d06b0fa07bb18d8202035e739a494ce760aa73129f60f4bf2bd22b407
Status: Downloaded newer image for centos:7
 ---> eeb6ee3f44bd
Step 2/7 : MAINTAINER Jitendra jitendra@pioneerbusineessolutions.in
 ---> Running in cc0920350806
Removing intermediate container cc0920350806
 ---> 71034dfbcc3f
```

Step3 – listing the custom-built image.

docker images

```
root@dockerdemo:~# docker images
REPOSITORY          TAG         IMAGE ID       CREATED          SIZE
test_centos         Dockerfile  92a2a12df8cc   20 minutes ago   910MB
custom-ubuntu       latest      292c55511acf   13 hours ago     192MB
dockerdemo-website  latest      03e0c3950141   19 hours ago     135MB
ubuntu              latest      a8780b506fa4   7 days ago       77.8MB
alpine              latest      9c6f07244728   3 months ago     5.54MB
hello-world         latest      feb5d9fea6a5   13 months ago    13.3kB
```

Step4 – running a new container using this custom container.

docker run -it --name cen_test_container test_centos:Dockerfile

```
root@dockerdemo:~# docker run -it --name cen_test_container test_centos:Dockerfile
[root@833293a52bb2 /]# rpm -q httpd git vim
httpd-2.4.6-97.el7.centos.5.x86_64
git-1.8.3.1-23.el7_8.x86_64
package vim is not installed
[root@833293a52bb2 /]# yum install -y vim
Loaded plugins: fastestmirror, ovl
Loading mirror speeds from cached hostfile
 * base: us.mirror.nsec.pt
 * extras: repos.lax.layerhost.com
 * updates: mirror.atl.genesisadaptive.com
Package 2:vim-enhanced-7.4.629-8.el7_9.x86_64 already installed and latest version
Nothing to do
[root@833293a52bb2 /]#
```

cen_test_container = new container name

test_centos:Dockerfile = passing existing docker image with ‘Dockerfile’ tag.

21. Pushing the docker image to docker hub

Friday, May 19, 2023 5:59 PM

Pushing the docker image to docker hub: - be ready with your custom username/image:tag

Step1 – login to docker hub using console

docker login

```
root@dockerdemo:~# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
e one.
Username: tomarj44
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Username: tomarj44

Password : XXX

Note: image must follow name as <docker hub username>/<image name>:<version>

Step2 – rename your image with appropriate convention:

docker tag test_centos:Dockerfile tomarj44/test_centos:Dockerfile

```
root@dockerdemo:~# docker tag test_centos:Dockerfile tomarj44/test_centos:Dockerfile
```

Verify:

```
root@dockerdemo:~# docker images
REPOSITORY              TAG         IMAGE ID       CREATED             SIZE
test_centos             Dockerfile  92a2a12df8cc   About an hour ago   910MB
tomarj44/test_centos    Dockerfile  92a2a12df8cc   About an hour ago   910MB
```

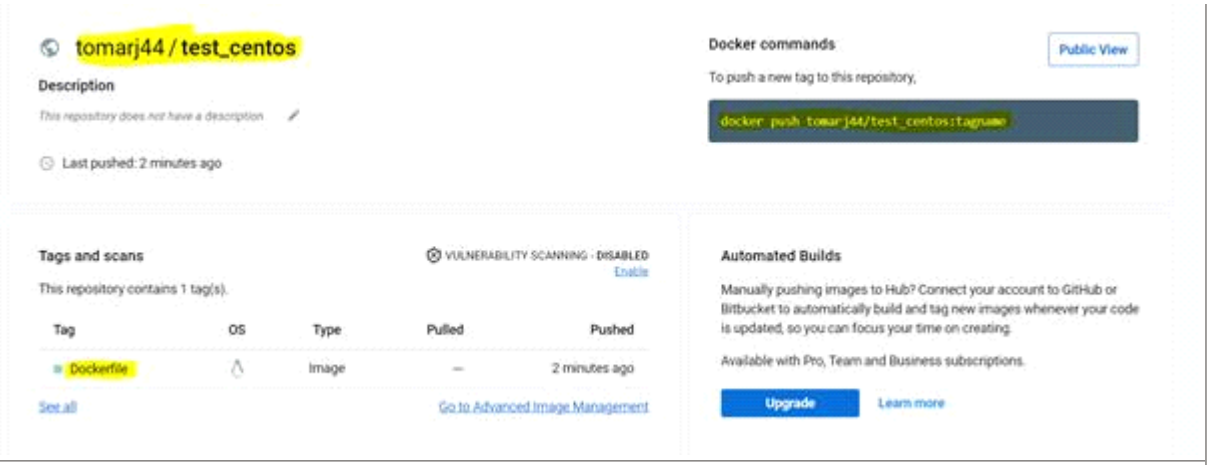
Step3 – Upload / Push your custom image on docker hub:

docker push tomarj44/test_centos:Dockerfile

```
root@dockerdemo:~# docker push tomarj44/test_centos:Dockerfile
The push refers to repository [docker.io/tomarj44/test_centos]
1cfe5bda793e: Pushed
fc577fe703f8: Pushed
a33817b811ed: Pushed
ae4fba6d1497: Pushed
b6c48e12cc2a: Pushed
174f56854903: Pushed
Dockerfile: digest: sha256:2cd51cb05d3591f747f63b956291ffae2fd82662db72882a6daf5582b66902c5 size: 1590
```

Step4 – Verify on docker hub:

Login to hub.docker.com



Step5 – download this image from another account

Login to VMWare W.S VM

docker pull tomarj44/test_centos:Dockerfile

```
[root@svr ~]# docker pull tomarj44/test_centos:Dockerfile
Dockerfile: Pulling from tomarj44/test_centos
2d473b07cdd5: Pull complete
553c605ca6a6: Pull complete
48caa1f7083b: Pull complete
1d378cb0610b: Pull complete
e85df4839226: Pull complete
5847aa9dbcb4: Pull complete
Digest: sha256:2cd51cb05d3591f747f63b956291ffae2fd82662db72882a6daf5582b66902c5
Status: Downloaded newer image for tomarj44/test_centos:Dockerfile
docker.io/tomarj44/test_centos:Dockerfile
[root@svr ~]#
[root@svr ~]# docker images
REPOSITORY              TAG          IMAGE ID       CREATED          SIZE
tomarj44/test_centos    Dockerfile   92a2a12df8cc   2 hours ago      910MB
hello-world             latest       feb5d9fea6a5   13 months ago    13.3kB
[root@svr ~]#
```

22. Testing App - 1st

Friday, May 19, 2023 6:02 PM

Dockerfile:

```
[root@docmaster docker-data]# cat Dockerfile
        # Fetching the image
        FROM centos:7

        # Setting up the working dir
        WORKDIR /app

        # Running the command
        RUN yum install -y httpd

        # Copying content to required directory
        COPY . /var/www/html/

        # Allowing port 80
        EXPOSE 80

        #running the HTTPD command
        CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Hello, World!</title>
   <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
   <div class="hello">
      Hello, World!
   </div>
<marquee>This app is running using Docker</marquee >
</body>
</html>
```

styles.css

```
.hello {
   color: blue;
   font-size: 24px;
   text-align: center;
   margin-top: 50px;
}
```

Note: once image is built & then later you change the code, it won't reflect even you restart

new containers as images are read-only copy. You need to re-build the image, if the code is changed.

23. Docker Networking Commands

Friday, May 19, 2023 6:10 PM

list existing networks
docker network ls

create a new network
docker network create <mynetwork>

verify if its created
docker network ls

Create and attach a new container to the network
docker run --network=mynetwork --name=mycontainer -d ubuntu:18.04

rename the container, if new container needs to be created.
docker run --network=mynetwork --name=mycontainer2 -d ubuntu:18.04

inspecting container
docker ps or (-a) //fetch container ID
docker inspect <container-id> //check last lines
docker inspect <container-id> | grep mynetwork
docker inspect d57b75e0097c | grep IPAddress

running container with interaction & specific network
docker run -it --network=mynetwork --name=mycontainer2 ubuntu:18.04

to connect with 2 different,
Start an nginx container, give it the name 'mynginx' and run in the background
docker run --name mynginx --detach nginx

Get the IP address of the container
docker inspect mynginx | grep IPAddress

Run busybox (a utility container). It will join the bridge network
docker run -it busybox sh

go inside bustbox & run below command:
/ # wget -q -O - 172.17.0.2:80
##
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
.
.
.
.
.

```
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
########################################
```

24. Docker storage commands

Friday, May 19, 2023 6:11 PM

listing all docker volumes

```
# docker volume ls
```

Create a new volume named 'my-vol'

```
# docker volume create my-vol
```

listing all docker volumes, again

```
# docker volume ls
```

inspecting volume

```
# docker volume inspect my-vol
        [
            {
                "CreatedAt": "2023-05-17T19:32:38+05:30",
                "Driver": "local",
                "Labels": null,
                "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
                "Name": "my-vol",
                "Options": null,
                "Scope": "local"
            }
        ]
        //The 'Mountpoint' is the location where the files or folders going to store
        actually
```

creating a file with some data onto this mount point

```
# cat > /var/lib/docker/volumes/my-vol/_data/demo.txt
        this is demo by
        Jeetu
```

attaching a single container to 2 containers & checking them.

```
# docker run -it -v my-vol:/root --name my-container1 alpine
        / # cd /root/
        ~ # ls
        demo.txt
        ~ # cat demo.txt
        this is a demo volume file.
        by
        Jeetu
```

creating a new container with the same volume attached & checking the same file.

```
# docker run -it -v my-vol:/root --name my-cont2 ubuntu
    root@4a419cc5e7c0:/# cd /root/
    root@4a419cc5e7c0:~# ls
    demo.txt
    root@4a419cc5e7c0:~# cat demo.txt
    this is a demo volume file.
    by
    Jeetu
```

25. Docker compose

Friday, May 19, 2023 6:11 PM

- Compose is a tool for defining and running multi-container Docker applications.
- Compose works in all environments: production, staging, development, testing, as well as CI workflows.
- It also has commands for managing the whole lifecycle of your application:
  - Start, stop, and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service

Installing standalone docker-compose version (link):

1. **To download and install Compose standalone,** run
   # curl -SL https://github.com/docker/compose/releases/download/v2.18.0/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
2. **Give executable permissions**:
   # chmod +x /usr/local/bin/docker-compose
3. **Verify:**
   # chmod +x /usr/local/bin/docker-compose
4. **Check/verify version**:
   # docker-compose version

To create a container using docker compose

1. Create a new folder (/root/docker-compose)
2. Change the directory to this folder
3. Create a file "**docker-compose.yaml**" (file name must be this only) & paste/write data given below:
   services:
     mytest:
       image: hello-world
   :wq!
4. Then, run below command:
   # docker-compose up

5. Valid the presence:
   # docker ps -a

26. Testing Docker-Compose-app

Friday, May 19, 2023 6:14 PM

Dockerfile content:	Index.php content	docker-compose.yml content

FROM php:7.4-apache
RUN docker-php-ext-install mysqli
COPY . /var/www/html/
	version: '3'
	services:
	  web:
	  build:
	  context: .
	  dockerfile: Dockerfile
	  ports:
	  - 80:80
	  depends_on:
	  - db
	  db:
	  image: mysql:5.7
	  ports:
	  - 3306:3306
	  environment:
	
	MYSQL_ROOT_PASSWORD: root
	  MYSQL_DATABASE: mydatabase
	  MYSQL_USER: myuser
	  MYSQL_PASSWORD: mypassword
		<?php
		$host = 'db';
		$user = 'myuser';
		$password = 'mypassword';
		$database = 'mydatabase';
		
		$conn = new mysqli($host, $user, $password, $database);
		
		if ($conn->connect_error) {
		  die('Connection failed: ' . $conn->connect_error);
		}
		
		$result = $conn->query('SELECT "Hello, MySQL!" AS message');
		$row = $result->fetch_assoc();
		$message = $row['message'];
		
		$conn->close();
		?>
		
		<!DOCTYPE html>
		<html>
		<head>
		  <title>PHP MySQL Example</title>
		</head>
		<body>
		 <?php echo $message; ?>
		</body>
		</html>

Run docker compose:

docker-compose up

In case, you re-edit the Dockerfile:

docker-compose build

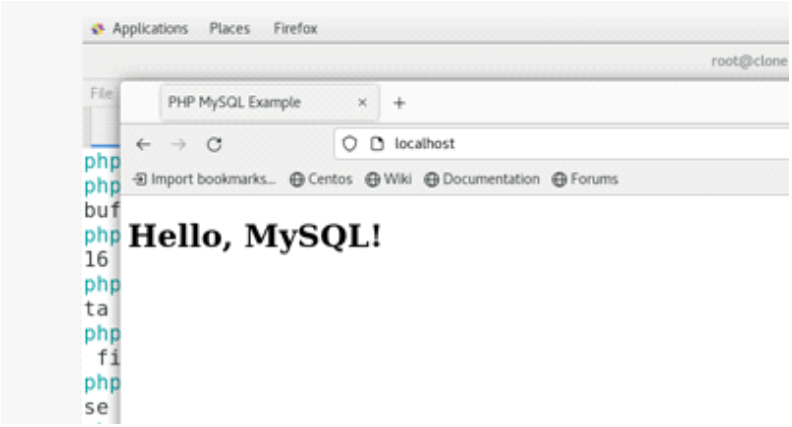
docker-compose up

To verify:

On web browser, http://localhost

To stop properly:

docker-compose down


Applications
Places
Firefox
root@clone
PHP MySQL Example
localhost
Import bookmarks...
Centos
Wiki
Documentation
Forums
Hello, MySQL!