**DOCKER COMPLETE**

Virtualization is the process of creating a software based virtual version of servers, apps, storage etc.

Virtualization is the process where one system is splits into many different sections, which was done by hypervisor.

**Diff Virtualization and Containerization**

* Containers provide an isolated environment for running the application. The entire user space is explicitly dedicated to the application.
* Any changes made inside the container is never reflected on the host or even other containers running on the same host.
* Containers are an abstraction of the application layer. Each container is a different application.
* Virtualization, hypervisors provide an entire virtual machine to the guest (including Kernel).
* Virtual machines are an abstraction of the hardware layer. Each VM is a physical machine.
* Docker is a platform for developers and sysadmins to build, run, and share applications with containers.
* Containerization is increasingly popular because containers are: Flexible, Lightweight, Portable, loosely coupled, Scalable, Secure, Short boot-up process.
* Container is nothing but a running process, which is encapsulated in order to keep it isolated from the host and from other containers.
* Each container interacts with its own private filesystem which is provided by a Docker image. An image includes everything needed to run an application.
* A container runs natively on Linux and shares the kernel of the host machine with other containers.

**DOCKER IMAGE:**

* An image is a text file with a set of pre-written commands, usually called as a Docker file
* Docker Images are made up of multiple layers which are read-only filesystem
* A layer is created for each instruction in a Docker file and placed on top of the previous layer
* When an image is turned into a container the Docker engine takes the image and adds a read-write filesystem on top.

**Docker Pull Working Process:**

First docker daemon look for an image in the local repo. If not found then it looks in docker registry. If found it pull the images and stores in local repo.

**CONTAINER - CONNECTION MODES:**

I-Interactive T- Connected to Terminal D-Detached Mode

* Detached Mode: $ docker run –itd ubuntu: xenial --->itd

1. user Manages from Daemon 2. Allow control over containers

* Root User Mode: $ docker run –it ubuntu: xenial --->it

1. User manages from root 2. Allow control over the containers of which user is attached.

**Docker run: Interactive Connected to Terminal**

▪ *docker run –it <image-name>*: This allows the docker container to run in the interactive mode as well as prevent it from exiting. Use ctrl+P+Q to get out of container without exiting

**Docker run: Detached Mode:**

*docker run –d <image-name>*: This allows the container to run a service in the background

* Containers started in detached mode exits when the root process used to run the container exits
* A container in detached mode cannot be automatically removed when it stops. Container stops when the root process used to run container stops.

**Docker Volume VS Bind Mount:**

**Volumes: (-v)** are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux).

* when you use a volume, a new directory is created within Docker’s storage directory on the host machine
* volumes stored in docker's "private" storage area
* volumes can be created by the docker engine when the container starts.

**Bind mounts: (--mount)**

* When you use a bind mount, a file or directory on the host machine is mounted into a container.
* Bind Mount can be stored anywhere
* Bind Mount have to exist prior to starting the docker container

**Docker Namespace:**

Namespace adds a layer of isolation in containers. Docker provides various namespaces in order to stay portable and not affect the underlying host system.

Few namespace types supported by Docker – PID, Mount, IPC, User, Network.

**Docker machine:** is a tool that lets you install Docker Engine on virtual hosts. Docker machine also lets you provision Docker Swarm Clusters.

**Copying Data to And from Containers**

$ *docker cp <container name>:<Source path> <Destination Path>*

**Commands Related to Docker File:**

* **FROM**: It shows where is the base image coming from.
* **MAINTAINER**: The section of the Docker file shows the maintainer or the owner of the Docker file.
* **RUN**: It is used to take the commands as its argument and runs it to form an image
* **CMD**: Command for starting up of a service of some kind. the actions to run when the containers are initiated is described in this section. Anything that is after a command is a list of things to run within any container that is initiated on a base image
* **ENV**: It is used to set one or more environment variables as $variable name or ${variable name}
* **WORKDIR**: It defines the location where the command defined by cmd is to be executed
* **VOL**: It is used to enable access from your container to a directory
* **ADD**: It copies the file into the containers own file system from the source on the host at the stated destination
* **EXPOSE**: It is used to expose the port to allow networking between the running process inside the container

**DIFF B/W RUN & CMD & ENTRYPOINT**

* RUN executes command(s) in a new layer and creates a new image. It is often used for installing software packages
* CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs
* ENTRYPOINT configures a container that will run as an executable
* CMD: Can be changed. Latest one will be executed
* ENTRY POINT: Can’t be changed

**ADD Vs COPY:**

**ADD**: Syntax: *ADD <source/URL> <destination>*

EX: ADD https://jdk 8. 0.. /usr/local/tomcat/webapps/java. War

-->ADD can copy from local to internet and also from internet to local

**COPY**: Syntax: COPY <source file> <destination>

EX: COPY Jenkins. War /usr/local/tomcat/webapps/Jenkins. War

---> COPY can only copy from local to internet

---> COPY don’t accept URL path.

**Importing and Exporting a Container to share with others:**

$ *docker export <container ID> > <name.tar>*

$ *docker import - <image name> <name.tar>* after this upload to webserver for download

$ *docker save -o <name.tar> <image name>* ----> To deal with committed images

$ *docker load < <name.tar>*

**NOTE: For pushing the image to docker hub add the image tag as <Docker username>/<image Name>**

**Docker Compose**: is a YAML file which contains details about the services, networks, and volumes for setting up the Docker application. So, you can use Docker Compose to create separate containers, host them and get them to communicate with each other.

**Docker Swarm**: serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

**Will you lose your data, when a docker container exists?**

No, you won’t lose any data when Docker container exits. Any data that your application writes to the container gets preserved on the disk until you explicitly delete the container.

**Monitor Docker in production:**

Docker provides functionalities like docker stats and docker events to monitor docker in production. Docker stats provides CPU and memory usage of the container. Docker events provide information about the activities taking place in the docker daemon.

**What changes are expected in your docker compose file while moving it to production?**

* Remove volume bindings, so the code stays inside the container and cannot be changed from outside the container.
* Binding to different ports on the host.
* Specify a restart policy
* Add extra services like log aggregator