Details of containers and images…

- A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.
- Containerized software will always run the same way, regardless of the environment.
- Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

Now comes Docker images: -

A docker image is an immutable file that is essentially a snapshot of a container or docker Image is the template (application plus required binaries and libraries) needed to build a running Docker Container (the running instance of that image).

Images are stored in a Docker registry such as registry.hub.docker.com

How can we build docker images? We have 2 ways to do so: -

- Update container created from an image and commit the results to a new image (Manual process)
- Create Dockerfile(Automated process)

- To turn an image into a container, the Docker engine takes the image, adds a read-write filesystem on top of it and initializes various settings including network ports, container name, ID and resource limits etc and then runs that image which we can say as container.
- A running container always has a currently executing process.
- A container can also be stopped (or exited in Docker's terminology), but an exited container is not the same as an image, as it can be restarted any time and it will retain its settings also we can modify any filesystem in container, but images are always immutable.
- A Docker image packs up the application and environment as per our project requirement to make all the applications to run but a container is a running instance of the images.
- Images are the packing part of docker, analogous to "source code" or a "program". Containers are the execution part of docker, analogous to a "process".
- Image is an equivalent to a class definition in OOPS and layers are different methods and properties of that class. Container is the actual instantiation of the image just like how an object is an instantiation or an instance of a class
- Container always uses common kernel of the OS on top of which Docker engine is installed and runs an docker image. A container is a self-sustainable app that will have packages and all the necessary dependencies together to run the code.

What Docker actually did??

- Downloaded the image from Hub / Registry
- Generated a new container
- Created a new file system

- Mounted a read/write layer
- Allocated network interface
- Setup IP
- Setup NAT
- Executed bash shell in container

Please do the practice of all below mentioned command, as I discussed all command separately in my previous video: -

docker container ps	→ list all running containers
docker container ps –a	→ list all running & dead containers
docker container commit	→ Create a new image from a container's changes
docker container cp	→ Copy files/folders between a container and the local filesystem
docker container create	→ Create a new container
docker container diff	→ Inspect changes to files or directories on a container's filesystem
docker container exec	→ Run a command in a running container
docker container export	→ Export a container's filesystem as a tar archive
docker container inspect	→ Display detailed information on one or more containers
docker container kill	→ Kill one or more running containers
docker container logs	→ Fetch the logs of a container
docker container ls	→ List containers
docker container pause	→ Pause all processes within one or more containers
docker container port	→ List port mappings or a specific mapping for the container
docker container prune	→ Remove all stopped containers
docker container rename	→ Rename a container
docker container restart	→ Restart one or more containers
docker container rm	→ Remove one or more containers
docker container run	→ Run a command in a new container

Till here we have covered what is docker images and container in depth: -

Now the question comes, can we build docker images as per our project requirement, if yes then how can we create customized docker images? There is two ways of doing so: -

Note: Images are created/saved with the build/commit commands, in manual creation of image we use commit command to save the modified container while in dockerfile we use build command to create a docker image.

1) **Manual Process** (commit command) (rarely used to create docker images)

- Take base image (always base layer of any of the container will always be any OS)
- start a container
- make changes to the container
- come out of container and make sure to use command ctrl+p+q, if you use exit command to come out, all the modification which we made inside the container will be killed or unsaved.
- save the container back as an image
- The **docker commit** command saves the running container into a new Docker images with all changes made.

2) **Automated Process – Using Dockerfile** (build command) (Mostly **used to create docker images**)

- Dockerfile: contains a set of docker instructions that provisions your operating system the way you like and installs/configure all your software's as per our requirement.
- Dockerfile is like your bash script that produce a tarball (Docker image)
- The **docker build** command builds an image from a dockerfile
- Use below mentioned link to get more concept on dockerfile: -
- https://docs.docker.com/engine/reference/builder/#usage