**Kernel primitives**

**Docker**

🡪First of all we are going to take a look at the stuff in the kernel that we use to build containers, and we are going to be doing Linux and windows. And

Then the other thing is the docker, the stuff that makes driving these kernel things nice and easy. We are going to kernel stuff first because that’s where the container building blocks live. Then we are going to do docker engine.

**Kernel:**

Container: Isolated area of an OS with resource usage limits applied.

Now to build then, we’ll leverage a bunch of low-level kernel stuff. In particular we use namespaces and control groups, and these fit in to below diagram

Control groups

Name Spaces-🡪Isolation area in OS

It is very hard to work with these Namespaces and control groups in the kernel.

And that’s why docker comes in to play.

The docker engine makes all of it’s easy.

The Docker engine is just like any other engine. So although we might interface with it through the CLI or directly to the API up here, a single endpoint, under the covers, it’s actually a bunch of smaller moving parts.

We use the command line to create a new container.

The client takes the command and makes the appropriate API request to

The containers/create endpoint here in the engine.

And the engine here pulls together all of

The required kernel stuff, and out pops a container.

**Def of Docker container/image:**

It only has the application layer of the OS and uses the kernel and CPU of the host machine. That’s why docker container boot’s very fast. In your host machine kernel is already running , So if you boot your docker container it will share the running kernel and start the container so fast.

**Use of kernel in docker**:

The docker technology uses the Linux kernel and features of the kernel, like Control groups and namespaces, to segregate (divide or set a part) processes so they can run independently.

**Def of kernel:**

It is the core part of an OS hence it has full control over everything in the system. Each operation of hardware and software is managed and administrated by the kernel.

**Namespaces and Control groups:**

Namespaces gives the isolation for the container with the underline host

Cgroups gives the ability to allocate the things to those containers.

**Docker Engine:**

In the docker at one side it exposes an API for us. and other side it interfaces with all the kernel magic. And out pop containers.

API

Docker pops out container

Namespaces Cgroups

Windows Linux

**Def:** Docker is a open source platform for developing, shipping, and running applications.

🡪Docker enables you to separate your applications from your infrastructure so you can deliver

Software quickly.

🡪Docker manages the lifecycle of the container.

🡪The use of containers to deploy applications is called containerization.

**How Docker works:** Staging area

Docker file🡪docker image🡪Docker Hub

Production server

Docker container

-----------------------------

Virtual machine(VM)

In docker file we will have the project code and it contains all the application dependencies. It is the recipe to create our image.

And then docker file is linked to the Docker image. Mainly the docker image is built using the docker file.

A Docker image is **a file used to execute code in a Docker container**

And then we have docker container it is the instance of the docker image. The docker container is used for to deploy the applications in to the docker hub.

The docker hub is the registry where the docker images are present.

**Note:** It was developed as an internal project at a platform-as-a service company called **DotCloud**

and later renamed as **Docker.**

**Working with Docker Container:**

Docker provides ability to package and run an application in an isolated environment called a container.

**Commands:**

docker container ls -a 🡪to list the containers

docker container create nginx 🡪to create new container

docker container create - -name web01 nginx🡪to name container as web01

docker container start web01🡪to start the container

docker container run - -name server01 centos🡪to create container using run

#docker container run - -name server02 -i-t centos

#ps

#exit 🡪to create a new container with interactive terminal, the container is created and interactive terminal is displayed. You can run the commands inside the container and exit.

#docker container run - -name server03 -dit centos🡪create a container with detached terminal

#docker container attach server03

#ps

Press ctrl+p,q🡪for attaching container with server03

#docker container inspect web02(container name) | grep -e “Hostport” -e “IPAdress”(ip address)

(Displays detailed information on one or more containers)

#curl ipaddress🡪let us access the webserver by using the container ip address.

#cat >index.html<<EOF

Welcome to docker!!

EOF

Docker container prune🡪to remove all stopped containers

**Docker image:**

Docker image ls🡪to list the images

Docker search image name🡪to search the image

Docker image pull image name🡪to pull the image

**Docker File:**

It is a text file with instructions to build image.

FROM

RUN

CMD

Step1: Create a file named docker file.

Step2: Add instructions in docker file.

Step3: Build docker file to create image.

Step4: Run image to create container.

RUN🡪It gets executed during the building of the image

CMD🡪It gets executed only when you create a container out of the image

**Commands:**

🡪login as root user

🡪create a directory and a docker file

#mkdir example && cd example

🡪create a docker file

🡪Take the docker file and use the docker build command to build an image

#docker image build -t centos:nginx

**Docker Hub:**

It is used to setup the local registry.

1. Signup with dockerhub.
2. Tag an image.
3. Push an image to dockerhub.
4. Clean up.

Docker Architecture:

Docker Client

Docker Daemon

Container Local images Image registry

🡪Docker Daemon is like a server component.

🡪Docker daemon is mainly responsible for the execution of that specific command.

And also it is responsible for managing containers, managing local images, pulling the image from the image repository.

We run the commands in Docker client and docker client sends it to docker daemon for execution. Whenever we try to run a container docker daemon checks the local images, is that specific image present locally? If it’s not present, docker daemon talks to image registry, gets the image down into our local images, and then runs it as a container.

Docker uses:

1. Standaridized application packaging 🡪 java, python, nodejs
2. Multi platform support🡪like aws, gcp, azure etc
3. Light weight and isolation.