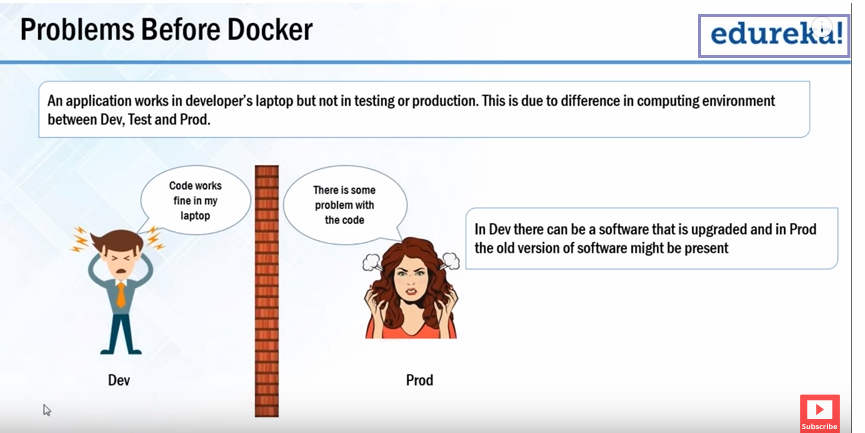
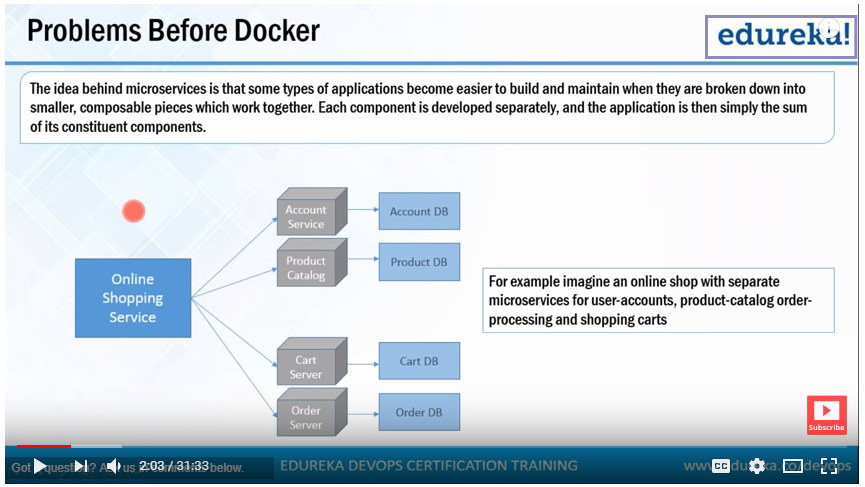
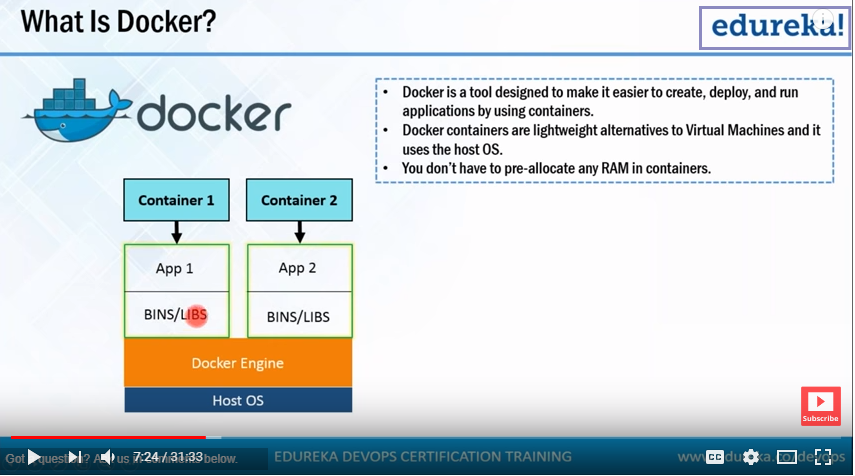
**Docker**

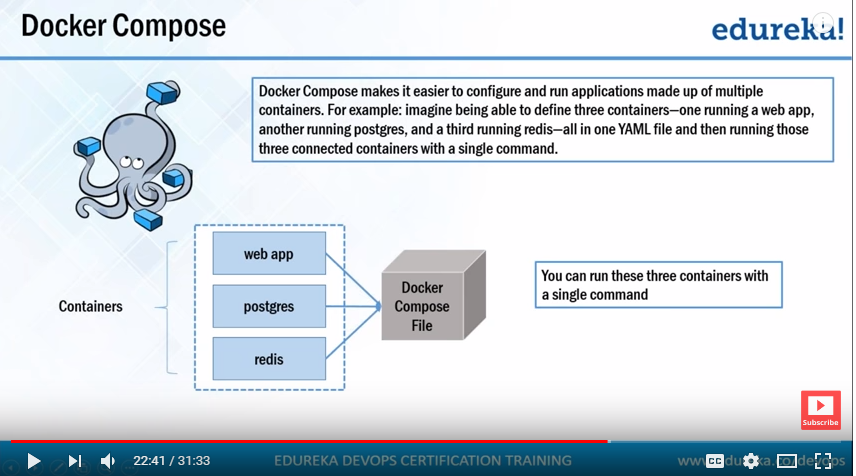
What is docker:

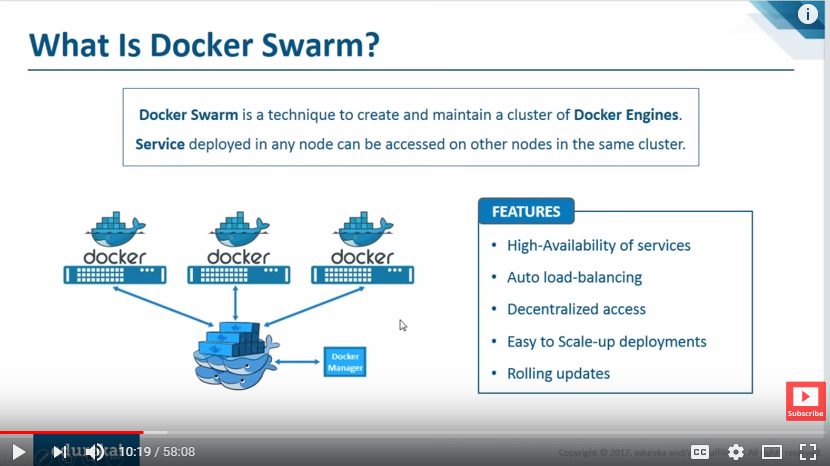
[Docker](https://github.com/docker/docker) is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.











# Docker

Image : Collection of Files, some meta data, read only file system.

Docker file specify the application requirements or dependencies in an easy to write docker file. that are the particular dependencies required the application all are present in docker file.

Container : Fundamental unit of packing. It is an encapsulated (Isolated) set of processes running in a read-write copy of a file system.

# Docker file:

FROM = image-name

RUN = This command executed at the time of Building (Image creation).

Add layers to images used to install apps.

CMD = This command executed at the time of container launch time

Equivalent of Docker run <arge> <command>

Equivalent of Docker run <arge> /bin/bash

ENTRY POINT = This command will be executed only one time. If you specify this command for

This is the last command of Docker launching

{{Default attribute specify Can’t be overridden at run time with normal commands at runtime, so we know where we have been typically specifying a command to run inside of ours, specify the arguments and it will execute at any condition.

# Docker Swarm

Manager Node = Dispatches the submitted tasks to worker nodes

Worker Node = Receive and execute tasks dispatched from manager nodes

Service =Definition of tasks to execute on worker nodes (Tasks, public hosts, copy of the tasks

running, volumes, replicas

Task = Carries a Docker container and the commands to run inside the container

Load balancing = Manager node used ingress load balancing to expose the services.



# Docker Commands:

docker run -it -v /test-v1 –name=volume-container Ubuntu:15.04 /bin.bash

# Mounting volume folder /test-v1 to volume-container

docker run -it --volume-from=”volume-container”-name Ubuntu:15.04 /bin.bash

# it will take same volume folder from volume container and copy all the data.

docker run -v /root/docker-engine/path : /root/container/tmp

# mounting volumes from engine to container

Docker info # Show Docker Engine Information.

Docker inspect <container name> # Show container information

#Delete Commands:

docker run -v <container-name> # delete the container volumes

docker images | grep “[hours/days/weeks/months]\* ago “ | awk ‘{print $3}’ | xarge docker rmi

# delete the images hours ago, days ago, weeks ago, months ago

docker rm -v $(docker ps -aq -f ‘status=exited’) # delete the stopped containers

docker rmi $(docker images -aq -f ‘danging=true’) # delete all untagged images <None:None>

docker rmi $(docker images -q) # delete all images

docker image prune –all, (--filter, -f ) # at least one used (image assigned to container) only this images will keep it & remaining all are deleted.

Docker (images, containers, volumes, networks) prune # delete all unused things

#Network Commands:

docker network ls # Show list

create # Create new network

connect #

disconnect # Disconnect network

inspect # Inspect the network

prune # Delete un-used network

rm # Delete network

# Create one net work

docker network create --subnet=10.1.0.0/16 \

--gatway=10.1.0.1 \

--ip-range=10.1.4.0/24 \

My\_own\_bridgenet \

# Create container with Own Network & IP

docker run -dti –name=container\_1 \

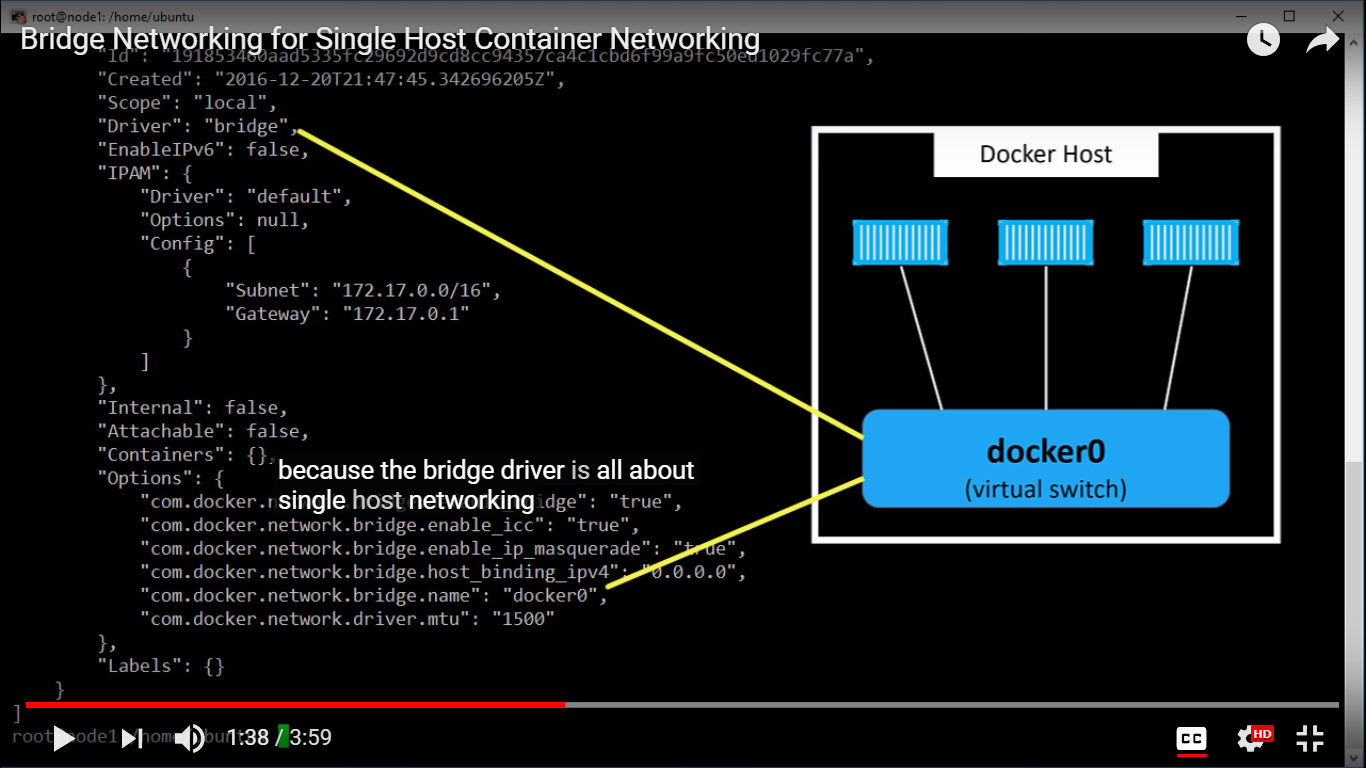
--network= My\_own\_bridgenet \

--ip=10.1.4.2 \

centos:latest \

/bin/bash

Docker host we have got a virtual switch or bridge called docker0 this is what really makes up that network called bridge well with that all we have to do is plumb containers into it and like any layer to type switch any containers



Install docker0 bridge

apt-get install bridge-utils

brctl show

What is Docker cache?

Whenever a container is created for the first time, its corresponding Dockerfile is also executed for the first time. This creates a new Docker image, which is stored in the folder ‘/var/lib/docker’ by default.

During further runs of Dockerfile, Docker will create and commit a new layer to the already existing image. These images are stored in the cache.

Unless explicitly instructed, Docker always looks for an existing image in its cache, whenever a new container is created. Reusing images from the cache helps to speed up the container deployment process in Docker.

Is Docker cache problematic?

While Docker cache helps speed up container creation process, there are certain situations where using the Docker cache will fail the purpose.

For instance, in the case of a source code checkout, we need the latest code to be available. In such cases, we can specifically give instructions not to use the cache.

This is referred to as busting the cache. The Dockerfile command in such cases would look like ‘docker build –no-cache=true’.

Another major concern is that the Docker images in the cache take up disk space. The size of a Docker image is the total space taken up by the image and all its parent images.

In the long run, you will find your Docker machine running out of disk space, as a result of all the Docker images here and there. So, its important to clear the cache regularly.

## What are dangling images?

Every Docker image has a repository name and tag associated with it. For instance, an Ubuntu Docker image may have ‘Ubuntu’ as the repository name and ‘latest’ as the tag.

As the name suggest, a dangling image in Docker is something that is inactive and not being used by any other running images.

The'repo:tag' for dangling images would be shown as <none>:<none> or untagged when the command ‘docker images’ is executed.

Since the dangling images cause wastage of disk space, they need to be deleted periodically to clear up the disk space for efficient functioning of the server

## How to remove dangling images

docker rmi $(docker images -q -f "dangling=true")

# Docker - How to cleanup (unused) resources

Once in a while, you may need to cleanup resources (containers, volumes, images, networks) ...

## delete volumes

// see: https://github.com/chadoe/docker-cleanup-volumes

$ docker volume rm $(docker volume ls -qf dangling=true)

$ docker volume ls -qf dangling=true | xargs -r docker volume rm

## delete networks

$ docker network ls

$ docker network ls | grep "bridge"

$ docker network rm $(docker network ls | grep "bridge" | awk '/ / { print $1 }')

## remove docker images

// see: http://stackoverflow.com/questions/32723111/how-to-remove-old-and-unused-docker-images

$ docker images

$ docker rmi $(docker images --filter "dangling=true" -q --no-trunc)

$ docker images | grep "none"

$ docker rmi $(docker images | grep "none" | awk '/ / { print $3 }')

## remove docker containers

// see: http://stackoverflow.com/questions/32723111/how-to-remove-old-and-unused-docker-images

$ docker ps

$ docker ps -a

$ docker rm $(docker ps -qa --no-trunc --filter "status=exited")

## Resize disk space for docker vm

$ docker-machine create --driver virtualbox --virtualbox-disk-size "40000" default