**Docker commands**

#docker image ls //listing the images

#docker container ls –a //to list all the images

**To pull the image to local**

#docker pull <image-name>

#docker pull centos

**To run the container**

#docker run –it <image name options> <image:version> /bin/bash

#docker run –it –name c1 centos:9 /bin/bash

#cat /etc/os-release //to check OS version

Ctl+P and Ctl+Q //to comeout of the container coming out of Background

#docker rm –f <containername> /<Container-ID> //remove the container forceibly

#docker run –it –p 9000:80 –name c1 centos:7 /bin/bash //port mapping

//installing apache in the container

#yum install httpd //installing apache

//start the apache

#/usr/sbin/httpd

//write the html file into the container

Cd /var/lib/html  
  
# vim index.html  
  
<h1> Hello Docker </h1>  
  
Get into the container

#docker attach c2

#docker exec –it c2 /bin/bash

Build own image

Step1 : write the dockerfile

From centos:7

ENV user=Vishnu

RUN yum install httpd –y

Copy index.html /var/www/html/

Expose 80

Step2: write into index.html

#wget yahoo.com

Build docker image

#docker build –t apache .

From : Base

LABEL: Department  
RUN: run the command inside the image os level  
CMD: Run the command installed software. It will override the command  
copy: copy host to image  
ADD: copy the zip to destination and extract

Entrypoint: Executes at the time of container starts  
  
volume  
USER: User to execute to particular user

WORKDIR: working directory  
  
Uploading Docker image to the Registery  
  
#docker tag apache:latest kashinath94/apache  
#docker push kashinath94/apache2

UN:XXX  
pwd:XXX  
  
It will push now

#docker volume ls

# cd /var/lib/docker/  
  
At no volume

#docker volume create myvol //created myvol

//create container

#docker run –it –name c1 –p 9000:80 –v myvol:/var/www/html centos:7 bash

Create text file

#pwd

Create touch file.txt

**Docker network Driver**

Bridge: default network driver. Container to container communicate

Host: to communicate host to container

Overlay: docker engine one vm to docker engine in other vm

Ipvlan: Maclan: using mac address it will communicate

**Build image with light weight**

Don’t use multiple run commands

Remove cache regularly

Multi stage building

**Docker Swarm**

It is Orchestration of docker container

If any one node is down it will be able to balanced the load

Service is application deployment in the directory

Docker in built in docker engine

Port no 2377

The network used in docker swarm is overlay network

Create two instances for swarm

To enable the swarm

#docker swarm

//intiilzation

#docker swarm init

//copy the docker join link in notepad

And run in the docker worker node

On worker node

# docker join link

This worker join the cluster

To manager node

#docker node ls

#docker service create –name s1 –p 9000:80 tutum/helloworld

#docker service ls

To check were service is running

#docker service ps s1

#curl ifconfig

#docker container ls

Ideal it runs in manager node

**To scale to 5 replicas**

#docker service scale s1=5

Fiver replicate

#docker service ps s1

2 one manager and 3 on worker node

On each refresh it uses internal load balancer to container

Docker machine

Realtime swarm is not used in production with less features

Disadvantages in Docker warm in production

**RUN vs CMD vs EntryPoint**

**RUN: for installing packages and applications**

**CMD:**

**Alphine vs Distroless Images**

Dark side of container. Have some vulnerabilities in the container

When Building Docker images. We’re Often concerned with twothings

1. The security
2. The size of the image

* The container image is composed of seversl Layers and each one of them could be a source of a vulnerability
* So, Scanning each and every layer is very crucial in Devsecops
* The smaller container images have a lesser chance to get exposed to potential vulnerabilities

Every instruction we put in the **Docker file** creates layer

The smaller the size the lesser the vulnerabilities

Better to use alphine based images

**BestPractices for Dockerfile**

1. Do not rum container as ROOT
2. Avoid copying unnecessary files. Use .dockerignore
3. Merge Layers
4. Using alphine or distroless images as base images
5. Using multistahe builds
6. Health Checks

7. Avoid exposing unnecessary ports

8. Hardcoding the credentials

9. Not all images are secure

10.Some images can be old. With unmaintained code and packages

**Multistage Buids**

**Alphine Images**

* Alphine Linux is linux distribution built around musl LIBc and BusyBox
* It is only 5 MB in size Which makes it a great image base for utilities and even production appications
* By using Alphine linux as the base image and adding only required dependencies/artifacts on top of it, results in smaller and cleaner docker image

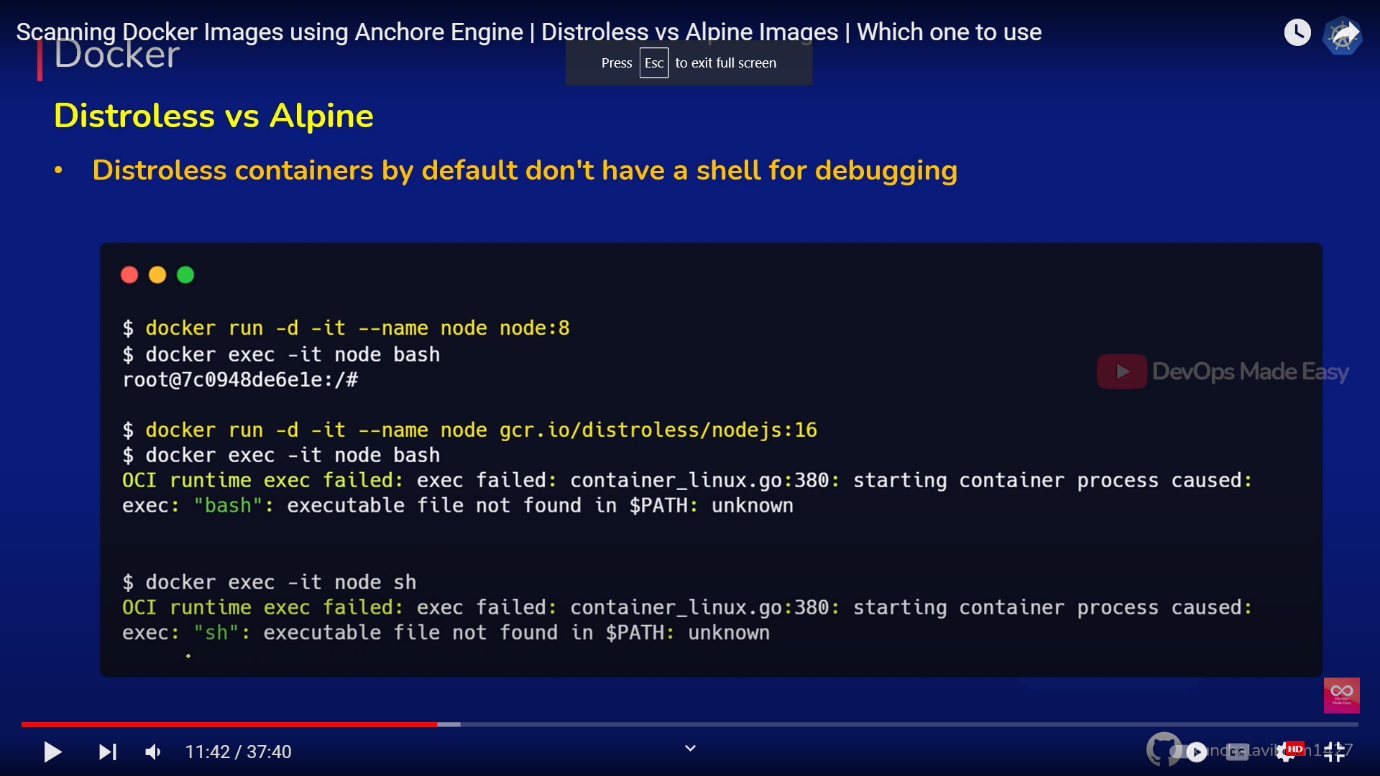
**Distroless container Images**

* Pioneered by google to improve security and container size
* Distroless Container Images are languages focused Docker Images,Sans the Operating System distribution
* Distroless Images only contains your application and its runtime dependencies, not other usual OS

**Benefits**

* Improves container security
* Reduces container size
* Reduces compliance scope
* Improves container distribution performances
* Cost-efficient by Utilizing lesser resources

Distroless containers by default don’t have a shell for debugging



**Container Security tools**

Scans the containers for all the vulnerabilities and monitor them regularly against any attack,issue and new bug

They Mostly work by scanning installed OS packages and comparing versions to the CVE(common vulnerabilities and Exposures) database

Some container Scanners

* Anchor Engine
* Clair
* Aquasecurity/Trivy
* Twistlock
* Qualys
* Sysdig Falco
* Blackdug
* Cilium

Command 1

>docker ps

Description : **to list existing docker containers in running state**

**Ps stands for “Process Status”**

**Command 2**

**>docker images**

**Description : to view the images**

**Command 3 : to run the imagesl**

**>docker run -itd --name nginx2 -p 81:81 nginx:latest**

**//Docker notes//**

Images: static binary files

Image: docker/getting started

**Command to run the application**

$docker run -d -p 80:80 docker/getting-started //port mapping

or  
$ docker run -dp 80:80 docker/getting-started //port mapping

Flag details  
  
-d - run the container in detached mode (in the background)

-p 80:80 - map port 80 of the host to port 80 in the container

docker/getting-started - the image to use

**Extended Description**

**Command**

**$ docker version [OPTIONS]**

**--format, -f :Format the output using the given Go Template**

**Get the server Version  
$** **docker version --format '{{.Server.Version}}'**

**or**

**$** **docker version -f '{{.Server.Version}}'  
  
Command2  
  
docker run**

**Usage**

**$** **docker run [OPTIONS] IMAGE [COMMAND] [ARG...]**

**Docker compose**

**Deploy multiple container in single file**

version: "3.7"

services:

app:

image: node:12-alpine

command: sh -c "yarn install && yarn run dev"

ports:

- 3000:3000

working\_dir: /app

volumes:

- ./:/app

environment:

MYSQL\_HOST: mysql

MYSQL\_USER: root

MYSQL\_PASSWORD: secret

MYSQL\_DB: todos

mysql:

image: mysql:5.7

volumes:

- todo-mysql-data:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: secret

MYSQL\_DATABASE: todos

volumes:

todo-mysql-data:

**Commands**

**$docker-compose up –d //up the file containers**

**$docker compose down //down the containers**