

Docker Def: Docker is the open-source centralised platform designed to create, deploy and run the application

- Docker is written in “go” language
- Docker uses container on the host O.S to run applications it allows applications to use the same linux kernel as a system on the host computer rather than creating a whole virtual O.S
- We can install docker on any O.S but docker engine runs natively on linux distribution.
- Docker is a tool that performs O.S level virtualization
- Docker was first released in March 2013; it is developed by Solomon Hykes and Sebastien Rahl

Docker commands:

- **Docker images** to see the images
- **Docker search Jenkins** to find out images in docker hub
- **Docker pull Jenkins** download image from docker hub
- **Docker run -it --name Sohail ubuntu /bin/bash** to give name to container
- **Service docker start** to start the docker
- **Service docker stop** to stop docker
- **Service docker status** to check status of the docker
- **docker rename (oldname)(new name)**
- **docker run <image_name>**
- **docker -d** To start the Docker daemon.
- **docker info** To display Docker system-wide information.
- **docker version** To display the current installed Docker version.
- **docker login -u <username>** To login into Docker.
- **docker push <username>/<image_name>** To publish an image to Docker Hub.
- **docker commit <container_name>** To create a new image from a container's changes.
- **docker history <container_name>** docker history
- **docker rmi <container_name>** To remove one or more images.
- **docker tag** To tag an image into a repository.
- **docker run --name <container_name> <image_name>** To create and run a container from an image.

- **docker run -p <host_port>:<container_port> <image_name>** To run a container with port mapping.
- **docker rm <container_name>** To remove a stopped container.
- **docker ps**
- **docker ps -a** To list all docker containers.
- **docker volume create** To create a new Docker volume.

To see all images present in your local machine
 C J# docker images

To find out images in docker hub
 C J# docker search jenkins

To download image from dockerhub to local machine
 C J# docker pull jenkins

To give name to Container
 → docker run -it --name bhupinder ubuntu /bin/bash
 interactive mode → terminal

To check, service is start or not
 → Service docker status

To Start Container
 → docker start bhupinder

To go inside container
 → docker attach bhupinder

To see all Containers
 → docker ps -a

To see only running Containers
 → docker ps ← Process status

To stop Container
 → docker stop bhupinder

To delete Container
 → docker rm bhupinder

Dockerfile

→ Dockerfile is basically a text file it contains some set of instruction

→ Automation of Docker image Creation

Docker Components

FROM → for base image This Command must be on top of the dockerfile

RUN → To execute Commands, it will create a layer in image

MAINTAINER → Author/ Owner/ Description

COPY → Copy files from local system (docker vm) we need to provide source, destination (We can't download file from internet and any remote repo)

ADD → Similar to COPY but, it provides a feature to download files from internet, also we extract file at docker image side

EXPOSE → To expose ports such as port 8080 for tomcat, port 80 for nginx etc

WORKDIR → To set working directory for a Container

CMD → Execute Commands but during Container Creation

ENTRYPOINT → Similar to CMD, but has higher priority over CMD, first commands will be executed by ENTRYPOINT only

ENV → Environment Variables

29:55 / 1:08:03 Try to Scroll for details

Dockerfile

- 1) → Create a file named Dockerfile
- 2) → Add instructions in Dockerfile
- 3) → Build dockerfile to Create image
- 4) → Run image to Create Container

① vi Dockerfile

② { FROM ubuntu

RUN echo "Technical guftgu" > /tmp/testfile

To Create image out of dockerfile

docker build -t myimg .

docker ps -a

docker images

Creating Volume from Dockerfile

- ① Create a Dockerfile and Write

```
{ FROM ubuntu  
  VOLUME ["/myvolume1"] }
```

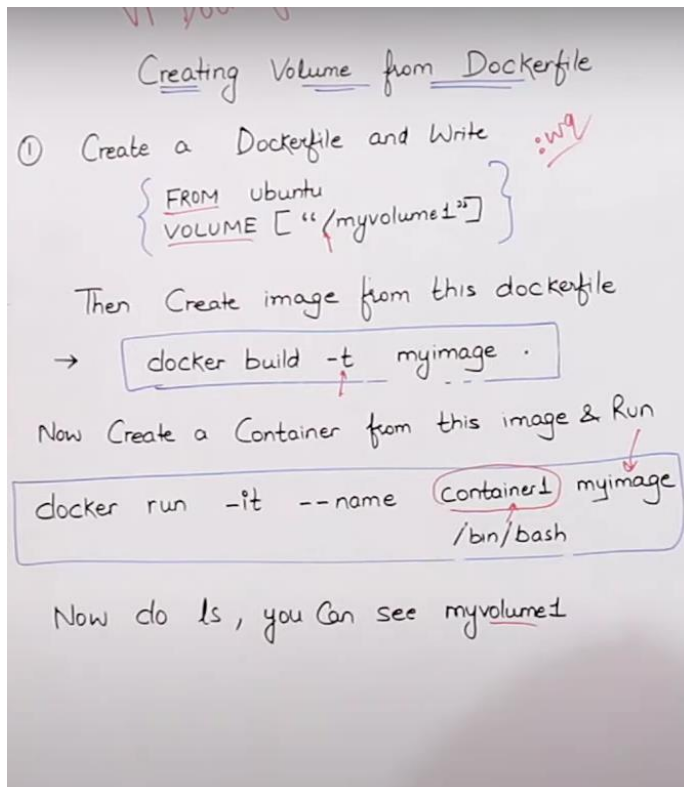
Then Create image from this dockerfile

→ docker build -t myimage .

Now Create a Container from this image & Run

docker run -it --name container1 myimage /bin/bash

Now do ls, you can see myvolume1



Docker Image: A Docker image is a file used to execute code in a Docker container .

Docker Container: A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application:

Deploying docker project using jenkins:

Install plugins

- CloudBees Docker Build and Publish plugin
- Docker API Plugin
- Docker Commons Plugin
- Docker Compose Build Step Plugin
- Docker Pipeline
- Docker plugin
- docker-build-step
- sonarscanner
- maven integration
- owsap dependency plugin
- jdk plugin

manage Jenkins configure the tools

Add Credentials for docker,git,sonarqube,

Create job

Select pipeline

Click ok

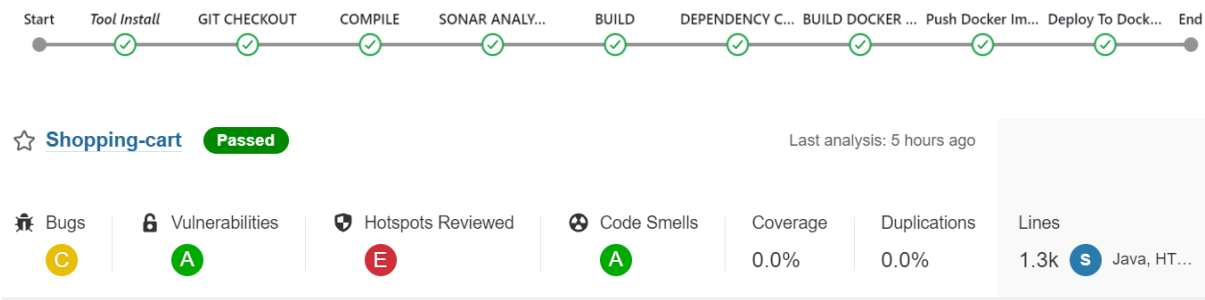
Pipeline select pipeline script

- pipeline {
- agent any
- tools{
- jdk 'jdk17'
- maven 'Maven'
- }
- environment{
- SCANNER_HOME=tool'Sonarqube'
- }
-
- stages {
- stage('GIT CHECKOUT') {
- steps {
- ○ git branch: 'main', url: 'https://github.com/GootySohail/Ekart.git'
- }
- }
- stage('COMPILE') {
- steps {
- ○ sh 'mvn compile'
- }
- }
- stage('SONAR ANALYSIS') {
- steps {
- ○ withSonarQubeEnv('Sonarqube') {
- ○ sh "\$SCANNER_HOME/bin/sonar-scanner \
- ▪ -Dsonar.projectKey=Shopping-cart \
- ▪ -Dsonar.sources=. \
- ▪ -Dsonar.java.binaries=. \
- ▪ -Dsonar.host.url=http://54.196.157.159:9000/ \
- ▪ -Dsonar.login=squ_2e950989af73a77e9c419c4c80d179783091e080
- "
- ○ }
- }
- }
- stage('BUILD') {
- steps {
- ○ sh "mvn clean package -DskipTests=true"
- }
- }
- stage('DEPENDENCY CHECK') {
- steps {
- ○ dependencyCheck additionalArguments: '--scan target/', odcInstallation: 'DC'
- ○ dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
- }
- }

- }
- stage('BUILD DOCKER IMAGE') {
- steps {
 - script{
 - withDockerRegistry(credentialsId: 'b19f8c77-88fb-45a2-9a3c-bd2326b75cac',
 toolName: 'docker'){
 - sh "docker build -t shopping-cart -f docker/Dockerfile ."
 - sh "docker tag shopping-cart gootysohail/shopping-cart:latest"
 - }
 - }
- }
- }
- stage('Push Docker Image') {
- steps {
 - script {
 - withDockerRegistry(credentialsId:'b19f8c77-88fb-45a2-9a3c-bd2326b75cac',
 toolName:'docker') {
 - sh "docker push gootysohail/shopping-cart:latest"
 - }
 - }
- }
- }
- stage('Deploy To Docker Container') {
- steps {
 - script {
 - withDockerRegistry(credentialsId:'b19f8c77-88fb-45a2-9a3c-bd2326b75cac',
 toolName:'docker') {
 - sh "docker run -d --name shopping -p 8070:8070 gootysohail/shopping-
 cart:latest"
 - }
 - }
- }
- }
- }
- }

Click apply and save

Click build now



After build success

Copy the ip and browse in browser

The figure shows a web application interface with a dark header bar. The header contains the text 'Shop' on the left and 'Registration' and 'Login' on the right. Below the header, there is a login form with two input fields: 'UserName' and 'Password'. A blue 'Login' button is positioned below the 'Password' field. The form is centered on the page.