Softwares installation Tutorials : https://ashokit.in/yt-content/softwares-installation

AWS Cloud Tutorials : https://ashokit.in/yt-content/cloud-tutorials

Linux tutorials : https://ashokit.in/yt-content/linux-tutorials

==========
Software Application Architecture
==========

1) Frontend : User Interface (UI)

2) Backend : Business logic

3) Database : Storage

==========
Tech Stack of Application
==========

Frontend : Angular 18v

Backend : Java 17v

Database : MySQL

Webserver : Tomcat

Note: If we want to run our application code, then we need to setup all required dependencies/softwares in the machine.

Note: dependencies nothing but the softwares which are required to run our application

==========
Application Environments
==========

=> In realtime we will use several environments to test our application.

1) DEV => 10 machines

2) SIT => 20 machines

3) UAT => 50 machines

4) PILOT => 100 machines

5) PROD (final delivery) => 200 machines

=> Dev env used by developers for code integration testing

=> SIT env used by Testers for system integration testing

=> UAT env used by client side team for acceptance testing (Go or No Go)

=> Pilot env used for pre-production testing

=> Prod env used for live deployment (end users can access our app)

=> As a devops enginner we are responsible to setup infrastructure to run our application

=> We need to install all required softwares (dependencies) to run our application

Note: We need to setup dependencies in all environments to run our application.

Note: There is a chance of doing mistakes in dependencies installation process (version compatability issues can occur)

==========
Life without Docker
==========

(1) "It works on my machine" problems

- Developers would build and test apps on their laptops.

- When they move the app to servers, it often breaks because of differences in OS, libraries, versions, configurations, etc.

Without Docker, teams would spend hours or days debugging these environment differences.

(2) Complex Software Installations

- installing something like a database (e.g., PostgreSQL) required manually:
    - download right version of s/w
    - install dependencies
    - configure services properly

(3) Heavy Virtual Machines (VMs)

- Before Docker, developers used Virtual Machines like VirtualBox, VMware, or Hyper-V to simulate servers.

- VMs are large, slow to start, and resource-hungry (each VM has a full OS inside).

(4) Scaling Applications is Difficult

- Manually configuring and copying app binaries.
- Manually setting up load balancers.

==========
What is Docker ?
==========

=> Docker is a free & open source software

=> Docker is used for containerization

Container = package (app code + required softwares)

=> With the help of docker, we can run our application in any machine very easily.

=> Docker will take care of dependencies installation required for app execution.

=> We can make our application portable using Docker.

==========
Docker Architecture
==========

1) Dockerfile

2) Docker Image

3) Docker Registry

4) Docker Container

=> Dockerfile is used to specify where is our app-code and what dependencies are required for our application execution.

=> Docker Image is a package which contains (app_code + dependencies)

Note: Dockerfile is required to build docker image.

=> Docker Registry is used to store Docker Images.

Note: When we run docker image then Docker container will be created. Docker container is a linux virtual machine.

=> Docker Container is used to run our application.

```
============================
Docker Setup in Linux VM
============================

Step-1 : Create EC2 VM (Amazon linux ami)

Step-2 : Connect with that vm using ssh client (git bash)

Step-3 : Execute below commands

# Install Docker
sudo yum update -y
sudo yum install docker -y
sudo service docker start

# Add ec2-user user to docker group
sudo usermod -aG docker ec2-user

# Exit from terminal and Connect again
exit

# Verify Docker installation
docker -v

=================
Docker Commands
=================

docker images : To display docker images available in our system

docker ps : To display running docker containers

docker ps -a : To display running + stopped containers

docker pull <image-id/name> : To download docker image from docker hub

        $ docker pull ashokit/spring-boot-rest-api

docker run <image-id/name> : To create/run docker container

        $ docker run hello-world

docker rm <container-id> : To delete docker container
```

```
docker stop <container-id> : To stop running docker container

docker start <container-id> : To start docker container which is in stopped state

docker rmi <image-id/name> : To delete docker image

docker logs <container-id> : To display container logs


# delete stopped containers + unused images + build cache
docker system prune -a

=====================================================
Running Real-world applications using docker images
=====================================================

### public docker image name (java springboot app) : ashokit/spring-boot-rest-api

docker pull ashokit/spring-boot-rest-api

docker run ashokit/spring-boot-rest-api

docker run -d ashokit/spring-boot-rest-api

Syntax : docker run -d -p <host-port>:<container-port> ashokit/spring-boot-rest-api

docker run -d -p 9090:9090 ashokit/spring-boot-rest-api

Note: To access application running in the container we will use below URL

## Java App URL : http://host-public-ip:host-port/welcome/{name}

Note: Host port number we need to enable in ec2-vm security group inbound rules to allow the traffic.


### public docker image name (python app) : ashokit/python-flask-app

docker pull ashokit/python-flask-app

docker run -d -p 5000:5000 ashokit/python-flask-app

## Python App URL : http://host-public-ip:host-port/

Note: Host port number we need to enable in ec2-vm security group inbound rules to allow the traffic.


========================
What is Port Mapping ?
========================

Note: By default, services running inside a Docker container are isolated and not accessible from
outside.

=> Docker port mapping is the process of linking container port to host machine port.

=> It is used to  allow external access to applications running inside the container.

Syntax : docker run -p <host_port>:<container_port> image_name

Note: host port and container port no need to be same.


=============
Dockerfile
=============
```

=> Dockerfile contains set of instructions to build docker image.

Filename : Dockerfile

=> To write dockerfile we will use below keywords

1) FROM
2) MAINTAINER
3) RUN
4) CMD
5) COPY
6) ADD
7) WORKDIR
8) EXPOSE
9) ENTRYPOINT

====
FROM
=====

=> It is used to specify base image required to run our application.

ex:

FROM openjdk:17

FROM python:3.3

FROM tomcat:9.0

FROM mysql:8.5

FROM node:19.5

============
MAINTAINER
============

=> MAINTAINER is used to specify who is author of this Dockerfile.

=> This is Optional in Dockerfile.

Ex : MAINTAINER Ashok <ashok.b@oracle.com>

=====
RUN
=====

=> RUN keyword is used to specify instructions (commands) to execute at the time of docker image creation.

Ex :

RUN 'git clone <repo-url>'

RUN 'mvn clean package'

Note: We can specify multiple RUN instructions in Dockerfile and all those will execute in sequential manner.

========
CMD
========

=> CMD keyword is used to specify instructions (commands) which are required to execute at the time of docker container creation.

Ex:

```
CMD 'java -jar app.jar'

CMD 'python app.py'
```

Note: If we write multiple CMD instructions in dockerfile, docker will execute only last CMD instruction.

=====
COPY
=====

=> COPY instruction is used to copy the files from source to destination.

Note: It is used to copy application code from host machine to container machine.

Source : HOST Machine

Destination : Container machine

Ex :

```
COPY target/app.jar  /usr/app/

COPY target/app.war /usr/bin/tomcat/webapps/

COPY app.py  /usr/app/
```

=====
ADD
=====

=> ADD instruction is used to copy the files from source to destination.

Ex :

```
ADD <source> <destination>

ADD <URL> <destination>
```

========
WORKDIR
========

=> WORKDIR instruction is used to set / change working directory in container machine.

ex:

```
COPY target/sbapp.jar /usr/app/

WORKDIR /usr/app/

CMD 'java -jar sbapp.jar'
```

========
EXPOSE
========

=> EXPOSE instruction is used to specify application is running on which PORT number.

Ex :

```
EXPOSE 8080
```

Note: By using EXPOSE keyword we can't change application port number. It is just to provide information the people who are reading our Dockerfile.

==========
ENTRYPOINT
==========

=> It is used to execute instruction when container is getting created.

Note: ENTRYPOINT is used as alternate for 'CMD' instructions.

```
CMD "java -jar app.jar"

ENTRYPOINT ["java", "-jar", "app.jar"]
```

==========
What is the diff between 'CMD' & 'ENTRYPOINT' ?
==========

=> CMD instructions we can override while creating docker container.

=> ENTRYPOINT instructions we can't override.

==========
Sample Dockerfile
==========

```
FROM ubuntu

MAINTAINER Ashok <ashok.b@oracle.com>

RUN echo 'hello from run instruction-1'
RUN echo 'hello from run instruction-2'

CMD echo 'hi from cmd-1'
CMD echo 'hi from cmd-2'
```

```
# create docker image using dockerfile
$ docker build -t img-1 .

# Run docker image to create docker container
$ docker run img-1
```

==========
Dockerizing java web app (Without SpringBoot)
==========

=> Java web app will be packaged as "war" file.

Note: To package java application we will use 'Maven' as build tool.

Note: war file will be created inside project "target" directory

=> To deploy war file we need web server (Ex: tomcat)

=> Inside tomcat server 'webapps' directory we need to place our war file to run the application.

```
######### Dockerfile to run java web app ############

FROM tomcat:latest

EXPOSE 8080

COPY target/app.war /usr/local/tomcat/webapps/
```

==========
Lab Task
==========

@@ Java Web App Git Repo : https://github.com/ashokitschool/maven-web-app.git

Note: Connect with Docker VM using SSH client and execute below commands

```
# install git client
$ sudo yum install git -y

# install maven s/w
$ sudo yum install maven -y

# clone project git repo
$ git clone https://github.com/ashokitschool/maven-web-app.git

# build maven project
$ cd maven-web-app
$ mvn clean package

# check project war file
$ ls -l target

# build docker image
$ docker build -t <img-name> .
$ docker images

# Create Docker Container
$ docker run -d -p 8080:8080 <image-name>
$ docker ps
```

=> Enable host port number in security group inbound rules and access our application.

URL :: http://public-ip:8080/maven-web-app/

==========
Dockerizing Java Spring Boot Application
==========

=> Every JAVA SpringBoot application will be packaged as "jar" file only.

Note: To package java application we will use 'Maven' as build tool.

=> To run spring boot application we need to execute jar file.

Syntax : java -jar <jar-file-name>

Note: When we run springboot application jar file then springboot will start tomcat server with 8080 port number (embedded tomcat server).

```
######### Dockerfile to run SpringBoot App ############

FROM openjdk:17
```

```
COPY target/app.jar /usr/app/

WORKDIR /usr/app/

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

==========
Lab Task
==========

Java Spring Boot App Git Repo : https://github.com/ashokitschool/spring-boot-docker-app.git

Note: Connect with Docker VM using SSH client and execute below commands

```
# clone project git repo
$ git clone https://github.com/ashokitschool/spring-boot-docker-app.git

# build maven project
$ cd spring-boot-docker-app
$ mvn clean package

# check project war file
$ ls -l target

# build docker image
$ docker build -t <img-name> .
$ docker images

# Create Docker Container
$ docker run -d -p 9090:8080 <image-name>
$ docker ps
```

=> Enable host port number in security group inbound rules and access our application.

URL :: http://public-ip:9090/

==========
Dockerizing Python application
==========

=> Python is a general purpose language.

Note: It is also called as scripting language.

=> We don't need any build tool for python applications.

=> We can run python application code directley like below

Syntax : $ python app.py

=> If we need any libraries for python (Ex: Flask) application development then we will mention them in "requirements.txt" file

Note: We will use "python pip" s/w to download libraries configured in requirements.txt file.

========== Python Flask App Dockerfile ==========

```
FROM python:3.6

COPY . /app/

WORKDIR /app/
```

```
RUN pip install -r requirements.txt

EXPOSE 5000

ENTRYPOINT ["python", "app.py"]

==========
Lab Task
==========

@@ Python Flask App Git Repo : https://github.com/ashokitschool/python-flask-docker-app.git


# Clone git repo
$ git clone https://github.com/ashokitschool/python-flask-docker-app.git

# Go inside project directory
$ cd python-flask-docker-app

# Create docker image
$ docker build -t ashokit/pyapp .
$ docker images

# Create container
$ docker run -d -p 5000:5000 pyapp
$ docker ps

# Access application in browser
URL : http://public-ip:5000/

========================
Assignments for today
========================

1) Setup Jenkins Server as Docker Container

2) Setup MySQL DB as Docker Container


================================
Dockerizing Angular application
================================

=> Angular is a framework which is used to develop Frontend applications (UI).

=> Angular project dependencies will be configured in "package.json" file

=> To build angular project we will use 'npm' as build tool. It is part of node software.

=> To run angular application we will use 'Nginx' as webserver.

=> Nginx server will run on HTTP protocol with 80 port number.

================= Angular app Dockerfile ======================

FROM node:18 AS build

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .
```

```
RUN npm run build --prod

FROM nginx:alpine

EXPOSE 80

COPY --from=build /app/dist/angular_docker_app /usr/share/nginx/html
```

```
=========
Lab Task
=========

Step-1 : Clone Git repo

$ git clone https://github.com/ashokitschool/angular_docker_app

Step-2 : Build Docker Image

$ cd angular_docker_app

$ docker build -t ngapp .

Step-3 : Create Docker Container

docker run -d -p 80:80 ngapp

Step-4 : Enable Http protocol with 80 port number in EC2 VM security group inbound rules.

Step-4 : Access our application in the browser

                URL : http://public-ip:80/

================================================================================

React JS App Git Hub repo : https://github.com/ashokitschool/ReactJS_Docker_App

================================================================================



============================================================
Q) How to push Docker Image to Docker Hub / Docker Registry ?
============================================================

$ docker images

$ docker tag <image-name> <docker-hub-username>/<img-name>:<tagname>

$ docker login

$ docker push <tagged-image-name>

========================================
Q) How to go inside docker container ?
========================================

$ docker exec -it <container-id> /bin/bash

================
Docker Network
================

=> Network is all about communication

=> Docker network is used to provide isolated network for containers
```

=> If we run 2 containers under same network then one contianer can communicate with another container.

=> By default we have 3 networks in Docker

1) bridge
2) host
3) none

=> Bridge network is used to run standalone containers. It will assign one IP for container. It is the default network for docker container.

=> Host network is also used to run standalone containers. This will not assign any ip for our container.

=> None means no network will be available.

=> We can use 2 other networks also in docker

1) Overlay
2) MacvLan

=> Overlay network is used for Orchestration purpose (Docker Swarm)

=> Macvlan network will assign physical Ip for our container.

```
# display docker networks
$ docker network ls

# create docker network
$ docker network create ashokit-nw

# inspect docker network
$ docker network inspect ashokit-nw

# create docker container with custom network
$ docker run -d -p 8080:8080 --network ashokit-nw sb-app-image

# delete docker network
$ docker network rm ashokit-nw
```

==========
Docker Compose
==========

=> Earlier ppl developed projects using Monolithic Architecture (everthing in single app)

=> Now a days projects are developing based on Microservices architecture.

=> Microservices means multiple backend apis will be avialable

Ex:
hotels-api
flights-api
trains-api
cabs-api...

=> For every API we need to create seperate container.

Note: When we have multiple containers like this management will become very difficult (create / stop / start)

=> To overcome these problems we will use Docker Compose.

```
=> Docker Compose is used to manage Multi - Container Based applications.

=> In docker compose, using single command we can create / stop / start multiple containers at a
time.

==================================
What is docker-compose.yml file ?
==================================

=> docker-compose.yml file is used to specify containers information.

=> The default file name is docker-compose.yml (we can change it).

=> docker-compose.yml file contains below 4 sections

version : It represents compose yml version

services: It represents containers information (image-name, port-mapping etc..)

networks: Represents docker network to run our containers

volumes: Represents containers storage location


======================
Docker Compose Setup
======================

# install docker compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -
s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Check docker compose is installed or not
$ docker-compose --version

================================================
Spring Boot with MySQL DB using Docker-Compose
================================================

version: "3"
services:
  application:
    image: spring-boot-mysql-app
    ports:
      - "8080:8080"
    networks:
      - springboot-db-net
    depends_on:
      - mysqldb
    volumes:
      - /data/springboot-app

  mysqldb:
    image: mysql:5.7
    networks:
      - springboot-db-net
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=sbms
    volumes:
      - /data/mysql
networks:
  springboot-db-net:
```

```
================================
 Application Execution Process
================================

# clone git repo
$ git clone https://github.com/ashokitschool/spring-boot-mysql-docker-compose.git

# go inside project directory
$ cd spring-boot-mysql-docker-compose

# build project using maven
$ mvn clean package

# build docker image
$ docker build -t spring-boot-mysql-app .

# check docker images
$ docker images

# create docker containers using docker-compose
$ docker-compose up -d

# check docker containers running
$ docker-compose ps

# stop docker containers running
$ docker-compose stop

# start docker containers running
$ docker-compose start

# delete docker containers using docker-compose
$ docker-compose down


===================================
Stateful Vs Stateless Containers
===================================

Stateless Container : Data will be deleted after container deletion.

Statefull Container : Data will be available permanently

Note: Docker containers are stateless by default.

Note: In spring-boot-mysql app, we are using mysqldb as docker container to store application data.
When we re-create containers db also got recreated hence we lost data (this is not accepted in
realtime).

=> To maintain data permanently, we need to make docker container as statefull.

=> To make container as statefull, we need to use Docker volumes concept.

================
Docker Volumes
================

=> Volumes are used to persist data which is generated by docker container.

=> Volumes are used to avoid data loss

=> Using volumes we can make container as statefull

=> We have 3 types of volumes in docker
```

```
                1) Anonymous volume (no name)
                2) Named Volume
                3) Bind mounts

# display docker volumes
$ docker volume ls

# docker volume create
$ docker volume create <vol-name>

# inspect docker volume
$ docker volume inspect <vol-name>

# Remove docker volume
$ docker volume rm <vol-name>

# Remove all docker volumes
$ docker system prune --volumes


================================
Making docker container stateful
================================

=> Create mount directory on host machine (path : /home/ec2-user/)

        $ mkdir app

=> Map this 'app' directory in docker-compose.yml like below

======================docker-compose.yml===============
version: "3"
services:
  application:
    image: spring-boot-mysql-app
    ports:
      - "8080:8080"
    networks:
      - springboot-db-net
    depends_on:
      - mysqldb
    volumes:
      - /data/springboot-app
  mysqldb:
    image: mysql:5.7
    networks:
      - springboot-db-net
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=sbms
    volumes:
      - .app:/var/lib/mysql
networks:
  springboot-db-net:
============================================================

=============
Docker Swarm
=============

=> It is an Orchestration platform

=> Orchestration means Managing the process (containers)

=> Docker swarm is used to setup docker cluster
```

```
=> Cluster means group of servers

============================
Docker Swarm Cluster Setup
============================

-> Create 3 EC2 instances (ubuntu) & install docker in all 3 instances using below 2 commands

$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh

Note: Enable 2377 port in security group for Swarm Cluster Communications


1  - Master Node
2  - Worker Nodes


-> Connect to Master Machine and execute below command

# Initialize docker swarm cluster
$ sudo docker swarm init --advertise-addr <private-ip-of-master-node>

Ex : $ sudo docker swarm init --advertise-addr 172.31.41.217

# Get Join token from master  (this token is used by workers to join with master)
$ sudo docker swarm join-token worker

Note: Copy the token and execute in all worker nodes with sudo permission

Ex: sudo docker swarm join --token SWMTKN-1-4pkn4fiwm09haue0v633s6snitq693p1h7d1774c8y0hfl9yz9-
8l7vptikm0x29shtkhn0ki8wz 172.31.37.100:2377

-> In Docker swarm we need to deploy our application as a service.

====================
Docker Swarm Service
====================

-> Service is collection of one or more containers of same image

-> There are 2 types of services in docker swarm

1) Replica (default mode)
2) global


$ sudo docker service create --name <serviceName> -p <hostPort>:<containerPort> <imageName>

Ex :  $ sudo docker service create --name java-web-app -p 8080:8080 ashokit/javawebapp

Note: By default 1 replica will be created


Note: We can access our application using below URL pattern

        URL : http://master-node-public-ip:8080/java-web-app/


# check the services created
$ sudo docker service ls

# we can scale docker service
$ docker service scale <serviceName>=<no.of.replicas>
```

```
# inspect docker service
$ sudo docker service inspect --pretty <service-name>

# see service details
$ sudo docker service ps <service-name>

# Remove one node from swarm cluster
$ sudo docker swarm leave

# remove docker service
$ sudo docker service rm <service-name>
```

```
================
Docker Summary
================

1) What is application architecture
2) Application Tech stack
3) Application Environements
4) Challenges in app deployment process
5) Containerization
6) Docker Introduction
7) Docker Architecture
8) Dockerfile Keywords
9) Docker Images
10) Docker Containers
11) Port Mapping & Detached Mode
12) Dockerizing Java Web App (war file)
13) Dockerizing Java Spring Boot App (jar file)
14) Dockerization Python App (.py file)
15) Docker Network
16) Docker Compose
17) Docker Swarm
```