**Docker & Kubernetes:**

**Pre-Requisites:**

**Basic Linux Commands**

**Part-1 : Docker**

**Part-2 Kubernetes**

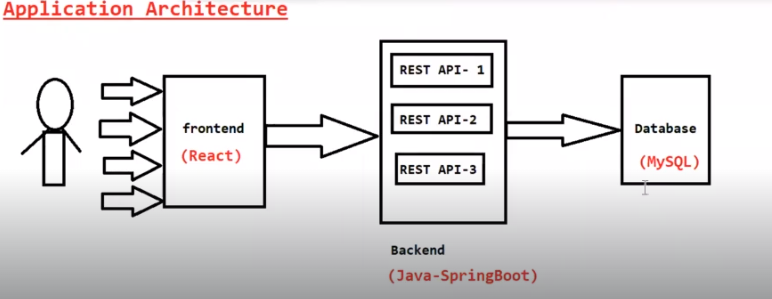
**Application Tech Stack:**

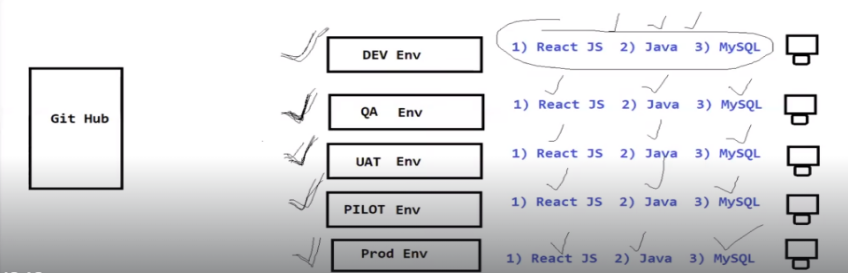
**It represents technologies used in the application**

**Front End: HTML,CSS,JS,BS & Angular / React JS.**

**Back End: Java/.Net/Python/Node JS.**

**Data Base: Oracle, MY SQL , Postgres SQL, Mongo DB.**





**Docker:**

* **Docker is Containerization Platform.**
* **Docker is used to simplify application deployment process in Multiple Environments**
  + **(DEV, SIT, UAT, PILOT, PROD)**
* **Docker is used to Package Application code + Application Dependencies for Easy Execution.**
* **Using Docker, we will create Docker Images**
* **Docker image Contains Code + dependencies**
* **We can run docker images in any machines. It will take care of Dependencies**
* **When we run the Docker Image . It will create a Docker Container.**
* **Docker Container will contain our Application**

**If we don’t use Docker deploying the application at the Multiple Environment, installing all the dependencies is very difficult tasks.**

**That’s Why we are Going for Docker**

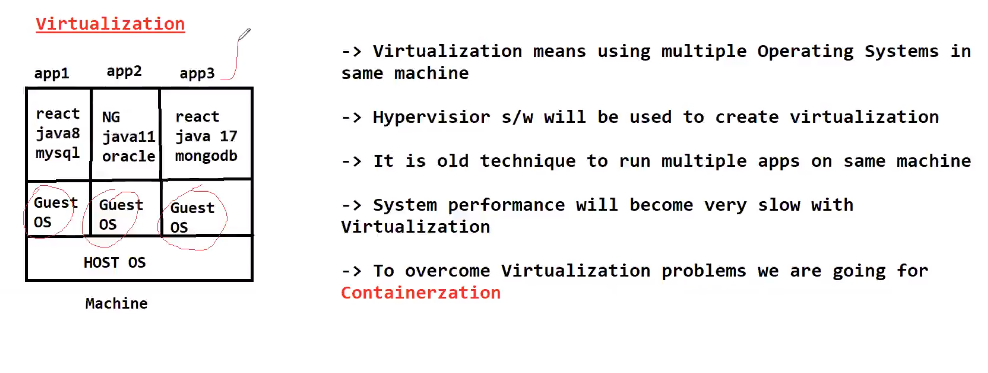


**Docker LOGO: The meaning of Log, which is easily shippable , Application can be shipped from one machine to another machine very easily.**

**Inside the container Application code+ Application dependencies is available.**

**Virtualization:**

* **Virtualization Means using Multiple Operation system in same Machine.**
* **Hypervisor s/w will be used to create virtualization.**
* **It is old Technique to run multiple apps on same machine**
* **System Performance will be very slow with virtualization**
* **To overcome problems virtualization Problem, we are going for Containerization.**

**Containerization:**

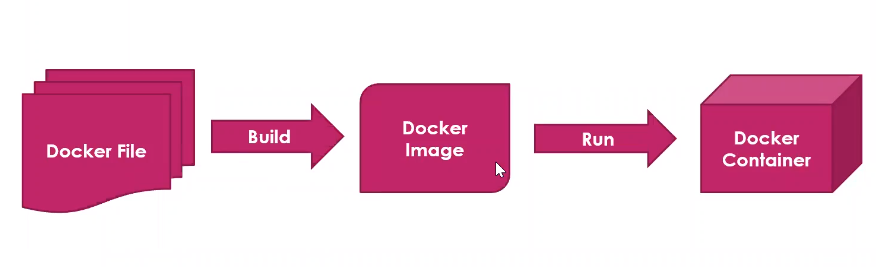
* **Docker is a containerization Software**
* **Using Docker will create Docker Image**
* **Docker Image => Application Code + Application Dependencies (java, Tomcat, Mysql)**
* **Once docker image is created then we can use Jenkins to run the docker image in multiple machines**
* **Jenkins is used to just deployment the software . We will use this to run docker images in all environments.**
* **When we run the Docker image it will create the docker container.**
* **Docker container mean run time instance of the application**

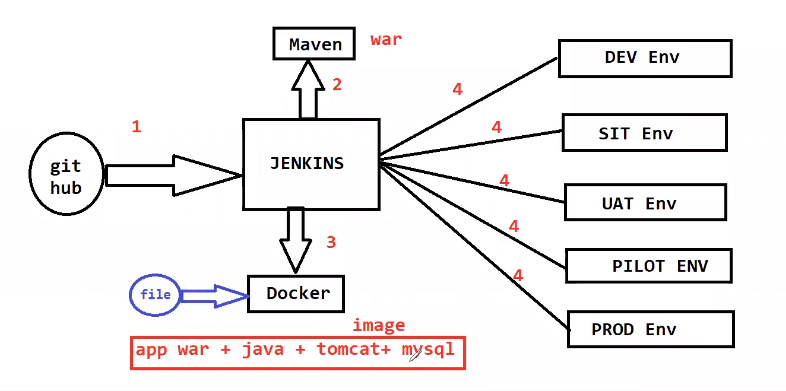
**Docker Architecture:**

* **Docker file contains set of instruction to build docker image. In docker file we will specify which software’s are required to run our code/application.**
* **Docker image Means package which contains Application code + Application Dependencies**
* **Docker Registry is a place which is used to store docker images for future purpose**

**Ex: Docker Hub, ECR etc ….**

* **Docker Container is run time instance of our application. When we run Docker image it will create Docker container. Inside Container our Application Code and Application dependencies will be available.**





**Demo:**

**Set up the Docker:**

**Install Docker in Amazon Linux**

**$ sudo yum update -y**

**$ sudo yum install docker -y**

**$ sudo service docker start**

**# add ec2-user to docker group by executing below command**

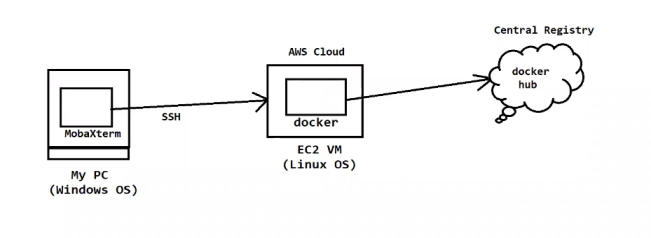
**$ sudo usermod -aG docker ec2-user**

**$ docker info**

**#Restart the session**

**$ exit**

**Then press 'R' to restart the session (This is in Mobaxterm)**



**Basic Docker commands**

**# Display docker images available in our machines**

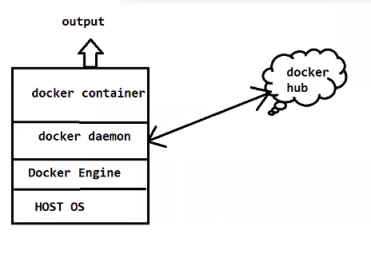
**$ docker images**

**# How to download the docker images**

**$ docker pull <imagename>**

**#Run Docker image**

**$ docker run <imagename>**



**# Display all running docker containers**

**$ docker ps**

**# Display all running and stopped Containers**

**$ docker ps -a**

**# To Delete the Container**

**$ docker rm <container ID>**

**# To delete the docker image**

**$ docker rmi <image name / image id>**

**# To delete the Docker image Forcefully**

**$ docker rmi -f <image name/image id>**

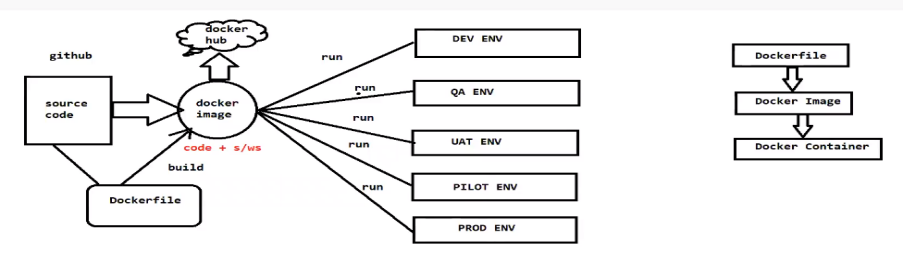
**Note: Hello-World it is very basic image. We use for understanding the basic commands**

**# stop the Docker Container**

**$ docker stop <container ID>**

**#Delete all stopped container and unused Image and un-used networks in a single shot.**

**$ docker system prune -a**



**Docker file:**

* **Docker file contains instruction to build the docker image**
* **In Docker file we will use DSL (Domain Specific Language) keywords**
* **Docker engine will process docker file instruction from top to bottom**
* **Below are the Docker file Keywords**

**FROM**

**MAINTAINER**

**COPY**

**ADD**

**RUN**

**CMD**

**ENTRYPOINT echo I am**

**ENV**

**LABEL**

**WORKDIR**

**EXPOSE**

**VOLUME**

**1.FROM: Keyword is used to represent base image to Create our image.**

**On top of base image our image will be created.**

**Syntax:**

**FROM java:jdk-1.8**

**FROM tomcate:9.5**

**FROM mysql:6.8**

**FROM Python**

**Which software are required for my application which is mentioned in base image.**

**Like this we can specify base image for our docker image**

**MAINTAINER: Maintainer is used to specify Docker file author information .**

**Who created the Docker file Developer or Devops**

**Syntax:**

**MAINTAINER Arun <official mail id>**

**2.COPY : Command is used to copy the files from source to destination while creating the docker image.**

**Syntax:**

**COPY <source-location> <destination-location>**

**Ex : target/sbi-app.war /app/tomcat/webapps/sbi-app.war**

**(war files is available in target folder)**

**Note: Code will represent the Copy Key Word.**

**3.ADD: Commands is also used Copy the files from source to destination while creating the docker images.**

**Syntax:**

**ADD <source-location> <destination-location>**

**ADD <url> <destination-location>**

**Ex :**

**ADD target/sbi-app.war /app/tomcat/webapps/sbi-app.war**

**ADD <URL> /app/tomcat/webapps/sbi-app.war**

**Q) What is difference between COPY and ADD Commands?**

**Both are using to copy the data from Source-Location-Destination-Location**

* **Using Copy Command, we can just copy the files from one path to another path within the machine.**
* **Using ADD Command , We can copy the files from one path to another path and it supports Source location as URL also.**

**4.RUN :**

* **RUN instruction will execute instruction while creating the image**
* **Using RUN Command, we can execute Shell Commands ( Linux Commands)**
* **We can write Multiple RUN instruction ; Docker will process all the run instructions from top to bottom**

**Example:**

**RUN yum install Maven**

**RUN yum install git**

**RUN git clone <url>**

**RUN mvn clean package**

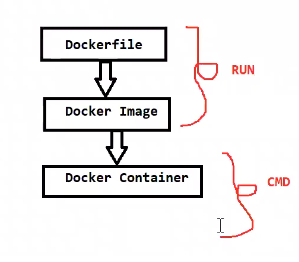
**5.CMD:**

* **CMD instruction will execute the instruction while creating the Container**
* **Using CMD We can run our Application package file jar/war file**

**Example:**

**CMD sudo start tomcat**

**Note: if you want to write the multiple CMD instruction also Docker file will process only last CMD instruction. There is no use of writing Multiple CMD instruction in one Docker file.**



**Q) What is difference between RUN and CMD in Docker file**

* **RUN is used to execute the instruction while creating the docker image**
* **CMD is used to Execute the instruction while creating the container.**
* **We can write multiple RUN instruction in Docker file ; Docker will process all those instructions one by one.**
* **If we write Multiple CMD instruction in Docker file, Docker will process only last CMD instruction.**

**Sample Docker file:**

**FROM ubuntu**

**MAINTAINER Arun**

**RUN echo “ Hi, I am RUN-1”**

**RUN echo “Hi , I am RUN-2”**

**RUN echo “Hi, I am CMD-1”**

**RUN echo “Hi, I am RUN-3”**

**RUN echo “Hi, I am CMD-2”**

* **Save the above content in docker file**

**File name : Dockerfile**

**# Command to create docker image using docker file**

**$ docker build -t <image name> .**

**t- tag name**

**. Current folder content**

**# Command to run the docker image**

**$ docker run <imagename>**

**# Command to login with docker hub account**

**$ docker login**

**Note: we need to enter our docker hub account credentials correctly ( it will ask first time)**

**# command to tag our docker image**

**$ docker tag <image name> <tag-name>**

**Ex: $ docker tag firstimage arunkeerthi3101/firstimage**

**# command to push docker image to dockerhub**

**$ docker push <tag-name>**

**Note: we can give customized name also for the docker file. When we changed docker file name we need to pass filename as input for docker build command using -f Option like below**

* **If you want to mention custom name for Docker file**
* **We need to Pass the custom name when we build the docker image.**

**Example:**

**Syntax: $ docker build -f <custom name> -t <tag-name> .**

**$ docker build -f <arun> -t <tag-name> .**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*6.ENTRY POINT**

**Entry Point instruction will execute while creating container.**

**Syntax:**

**ENTRYPOINT [“echo”, “welcome to arunit”]**

**ENTRYPOINT [“java”,”-jar”,”target/boot-app.jar”] -> for spring boot application**

**Q) what is difference between CMD and ENTRYPOINT**

**We can override the CMD instruction in runtime while creating the container**

**We cannot override ENTRYPOINT instructions.**

**$docker run <image> --CMD mkdir**

**7.WORKDIR**

**It is used to set working directory for an image/container**

**EX:**

**WORKDIR /app/**

**Note: The Docker file instruction which are available after WORKDIR those instruction will be processed from given working directory.**

**8.ENV:**

**ENV is used to set Environment variables.**

**Ex:**

**ENV <key> <value>**

**ENV <java> /etc/software/java**

**9.ARG**

**It is used to remove Hard-coded values**

**Using ARG we can pass values in the run time like below**

**Whenever we are creating the image, we can supply dynamic value to the arg**

**Ex:**

**ARG branch**

**RUN git clone -b $branch <repo-url>**

**$ docker build -t imageone –build-arg branch=master**

**10.USER**

**We can set user for creating image or container**

**With which user**

**After USER instruction all the remaining commands will execute with user permissions**

**11.EXPOSE:**

**It used to specify our container running PORT**

**Whatever port number container using we will specify that in docker file by using EXPOSE.**

**Ex:**

**EXPOSE 8080**

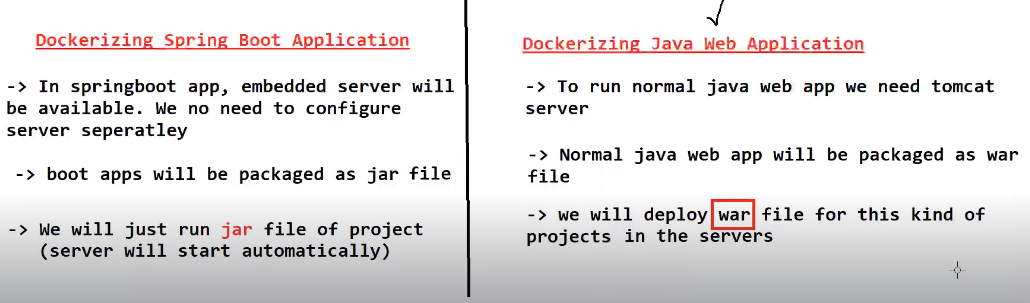
**Note: It is just like documents command to provide container running port number**

**Volume:**

**Volume is used to specify docker container data storage location**

**Note: Volumes are used for storage**

**If you want to Dockerizied the Application we should know how to write the Docker file.**



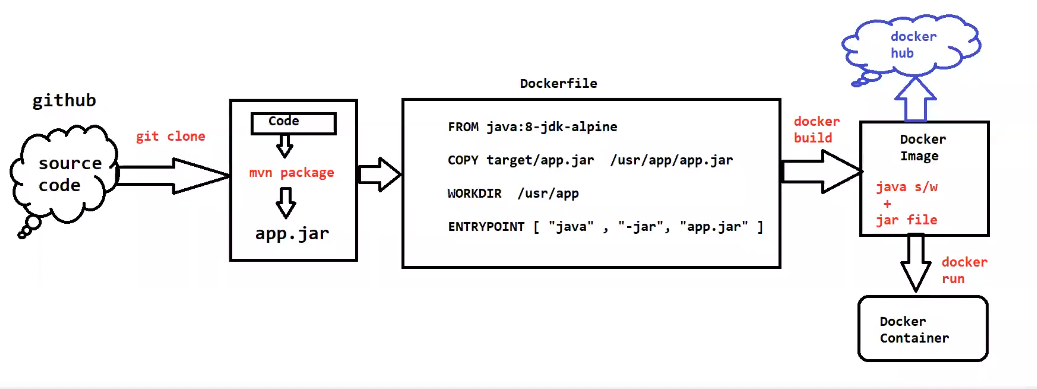
**Dockerize the Spring Boot Application:**

* **Spring Boot is one readymade java-based framework available in the market to develop java-based application quickly**
* **Spring boot is providing embedded server ( Internal server will be available, we no need to configure server separately for execution)Boot will take care of the server .**
* **Spring Boot Application will be packaged as jar file (mvn clean package goal will do that)**

**Note : when we do maven package , project jar will be created in project target folder**

* **To Run spring boot application, we just need to run packaged jar file like below**

**$ java -jar <boot-app-jar-file>**



**Setup Process Dockerizing Spring Boot Application :**

**Docker file:**

**FROM openjdk:11**

**COPY target/spring-boot-docker-app.jar /usr/app/**

**WORKDIR /usr/app/**

**ENTRYPOINT ["java", "-jar", "spring-boot-docker-app.jar"]**

**#install git client software**

**$ sudo yum install git -y**

**#Clone Git Repo <https://github.com/arunkeerthi/spring-boot-docker-app.git>**

**$ git clone <url>**

**# Navigating into project folder**

**$ cd <git clone project name>**

**# install maven s/w**

**$ sudo yum install maven**

**# Execute Maven Goals**

**$ mvn clean package**

**Note: After package got success, we can see project jar in target folder**

**# create docker image**

**$ docker build -t <imagename> .**

**# Run the docker image with port mapping**

**$ docker run -p “8080:8080” <imagename>**

**Note: Enable 8080 Port number in EC2 VM Security Group**

**URL To access Application : public ip:port number**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**How to run the container in detached mode.**

**=>with below command our terminal will be blocked . we can’t execute any other command. To execute other command, we need to type CTRL+C then terminal will open for commands execution but our container get stopped**

**$ docker run -p 8080:8080 arun/app**

**Note: To overcome above problem we can Pass “-d” to run container in detached mode. When we execute below command it will run the container in detached mode and it will open terminal for commands execution.**

**$ docker run -d -p 8080:8080 arunit**

**Once above command is execute, we can see running container using below command**

**$docker ps**

**We can check the docker logs of the container using below command**

**$ docker logs <container id>**

**Dockerizied the java Application (without spring boot)**

**Java Web application will be packaged as war file**

**WAR (Web Archive ) Container Application Code**

**To run the War file, we need webserver (Ex: Apache Tomcat)**

**We need to Deploy war file in Tomcat Server for Execution**

**In Tomcat server will have “web apps” folder for deployment**

**Note: To run Java Web Application we need “java & Tomcat” as dependencies.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Docker file\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FROM tomcat:8.0.20-jre8**

**COPY target/01-maven-web-app\*.war /usr/local/tomcat/webapps/maven-web-app.war**

**We can write Expose also 8080**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Docker file Link: <https://github.com/arunkeerthi/maven-web-app.git>**

**Git clone <https://github.com/arunkeerthi/maven-web-app.git>**

**$ cd maven-web-app**

**$ mvn clean package**

**$ docker build -t maven-web-app**

**$ docker images**

**$ docker run -d -p 8081:80 maven-web-app**

**$ docker ps**

**$ docker log <container id>**

**Context path nothing but your war file name.**

**Note: In the above url “maven-web-app” is called as context path (name of the war file will become context path)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Dockerize Python Application**

**Python is a general-purpose Scripting languages**

**Python will have .py as extension**

**Compilation is not required for python programs**

**\*\*\*\*\*\*\*\*\*\*\*\*\*Docker file\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FROM python:3.7**

**WORKDIR /usr/app**

**COPY . .**

**RUN pip install –no-cache-dir -r requirements.txt**

**EXPOSE 5000**

**CMD [“python3”,”flask”,”run”,--host:0.0.0.0”]**

Python flask app Git Hub Repo **<https://github.com/arunkeerthi/python-flask-docker-app.git>**

**$ git clone <https://github.com/arunkeerthi/python-flask-docker-app.git>**

**$ cd python-flask-docker-app**

**$ docker build -t python-flask-app**

**$ docker images**

**$ docker run -d -p 5000:5000 python-flask-app**

**$ docker ps**

**$ docker logs <container id>**

**# Command to enter into docker container**

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

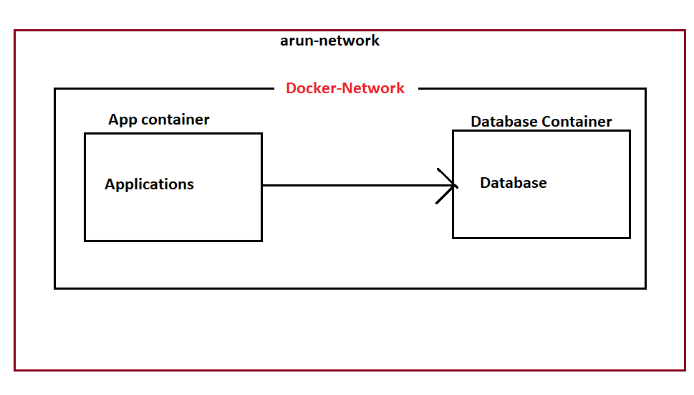
**$ cat /etc/os-release (OS of the container)**

**$ uname -r ( kernel of the container)**

**Note: Execute “exit” Command to come out from docker container VM.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Docker Network:**

****

**Network is all about communication**

**Docker Network is used to provide isolated Network for Docker Container**

**In Docker we will have below 3 default networks**

* **None**
* **Host**
* **Bridge**

**In docker we have below 5 networks drivers**

* **Bridge -> This is the default network in Docker**
* **Host**
* **None**
* **Overlay** 🡪 **Used for Docker Swarm**
* **Macvlan**
* **Bridge driver is recommended driver when we are running standalone container . It will assign one IP for container**
* **Host Driver is also used for standalone container . IP will not be assigned for container**
* **None Means no network will be provided by our Docker Containers.**
* **Overlay network driver is used for Orchestration . Docker Swarm will use this Overlay Network driver .**
* **Macvlan driver will assign MAC address for a container. It makes our container as Physical.**

**#display docker network available**

**$ docker network ls**

**#create docker network**

**$ docker network create <name>**

**#inspect network**

**$docker network inspect <network-id>**

**#Delete docker network**

**$ docker network rm <name>**

**# create container with you own network**

**$ docker run -d -P –network <Network name/id> <image name>**

**Note: If two containers are communicated together, both networks should be running in same network.**

**Docker Compose:**

A diagram of a computer

Description automatically generated

**Monolythic Application: One Application/Project Which Contains all the functionalities is called as monolithic Application**

* **If we make any small code change then we need to re-deploy entire application**
* **If we make any small code change in one functionality there may be an impact another functionality**
* **Maintenance will become very difficult when we go for Monolithic Based applications**

**Note: To Overcome the Problems of monolithic we are using Micro Service architecture**

**Microservice Application: Application functionalities will be developed as Micro Service (rest Apis)**

**Every functionality will be developed as individual Project (individual API)**

**ADMIN\_API**

**REPORT\_API**

**PAYMENT\_API**

**REPORT\_API**

**Microservice as Multi-containers-based Application.**

**Every API should run in Separate Container**

**Running Multiple Containers Manually for all the api’s is difficult jobs**

**To Overcome the Problem Docker compos comes into the picture.**

**Docker Compose Tool:**

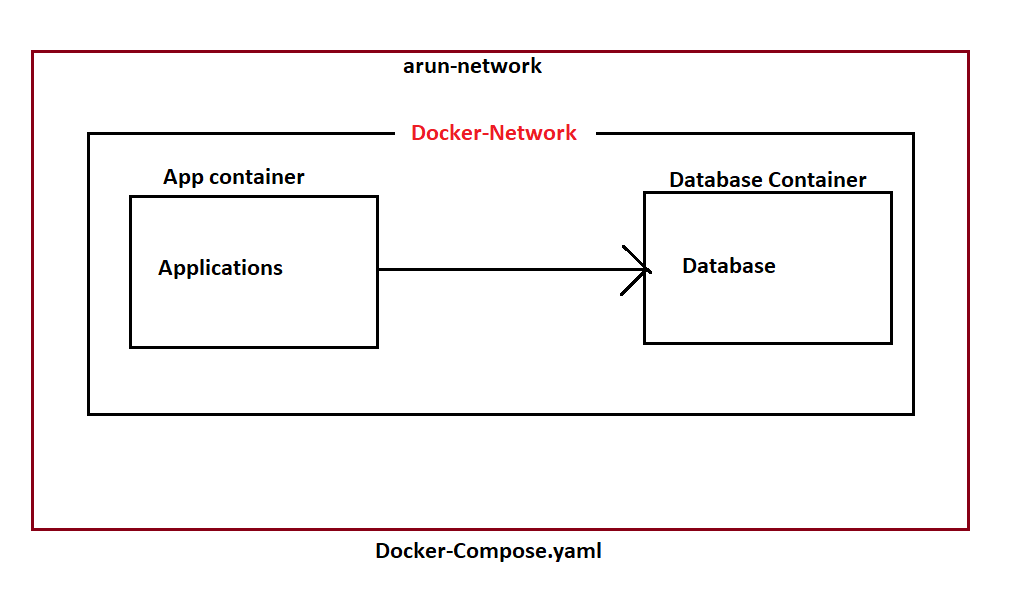
* **Docker Compose is a tool which is used to manage multi-Container based Applications**
* **Using Docker Compose we can easily setup and deploy multi-Container based Applications**
* **We will give containers information to Docker Compose using YML File (docker-compose.yaml)**
* **Docker Compose YML Should have all the information related to containers creation.**

**Note: Instead of managing the container manually , we are using docker compose to manage this container.**

**Docker Compose YML file:**

* **Docker Compose contains 4 sections Mainly**

1. **Version: (which version of docker compose)**
2. **Services: (nothing but container , How many containers do we want to create)**
3. **Network:**
4. **Volumes:**

****

**Docker Compose default file name is “docker-compose.yaml”**

**#Create Containers Using Docker Compose**

**$ docker-compose up**

**#Create Containers using Docker compose with custom file name**

**$ docker-compose -f <filename> up**

**#Display Containers created by Docker Compose**

**$ docker-compose ps**

**# Display docker compose images**

**$ docker-compose images**

**#stop & remove the container created by docker compose**

**$ docker-compose down**

**Docker Compose Setup**

**# download 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**

**# Give permission**

**$ sudo chmod +x /usr/local/bin/docker-compose**

**# How to check docker compose is installed or not**

**$ docker-compose --version**

A screenshot of a computer program

Description automatically generated

**Spring Boot with MySQL using Docker Compose:**

A diagram of a network

Description automatically generated

**Interview: Q:They will ask that how you will connect App to Database?**

**A:By using Database Source Properties .**

**Q:Where will we configure the Database Properties ?**

**A: We will configure database Source properties in Application.yaml**

* **Spring Boot App with MySQL DB Git repo URL**

**URL:** <https://github.com/arunkeerthi/spring-boot-with-compose-new.git>

* **Below is the docker-compose file**

**Step to Run Spring Boot Application with MySQL using Docker Compose:**

**# Clone git repository URL**

**$** <https://github.com/arunkeerthi/spring-boot-with-compose-new.git>

**# Get into Project Directory**

**$ cd spring-boot-mysql-docker-compose**

**# Build Maven Project**

**$ mvn clean package**

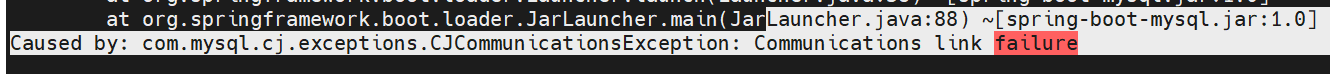
**# create Docker Image: (image name ‘spring-boot-mysql-app)**

**$ docker build -t spring-boot-mysql-app**

**#check docker images create or not**

**$ docker images**

**Note: if we run this application image alone as container (our application not connected to the database)**



**# Run container using docker compose ( docker-compose.yml available)**

**$ docker-compose up -d**

**#check container which are created**

**$docker-compose ps**

**#check logs of application container**

**$ docker logs <container-name>**

**Note: Access the Application in Browser**

**URL: https//ec2-vm-public-ip-host-port**

**# Get into APP container to check jar file**

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

**# check DB tables by entering into Container**

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

**$ mysql -u root -p (client command)**

**$ show databases;**

**$ use <db name>; (db name is sbms)**

**$ show tables;**

**$ select \* from <table-name> (our table name is book)**

**Example:**

**$ select \* from book.**

**What are the problems if we configure the Database\_source\_properties in Application .Yaml ?**

1. **Database source Properties credentials are visible over public.**
2. **If we want to deploy the different and different environments every time, we need to change the database properties credentials , build the image , deploy the image because tightly-Coupling**
3. **Maintenance problem will occur . we should not configuration properties within the application, always we need to separate project and project properties.**
4. **But in real time we are not doing like this , to overcome this problem . we are using the concept of called config map and secretes in Kubernetes to overcome tightly coupling issues.**
5. **By using config map and secretes will supply the data base properties dynamically.**

**Conclusion of Docker compose: Instead of managing the container manually , we are using docker compose to manage the multi-container.**

**Stateful Containers VS Stateless Containers:**

**Stateless Containers means container will not remember the data which got generated by that container. When we recreate the new container, we will lose old data.**

**Note: By default, docker container are stateless containers .**

* **In above spring boot application when we recreate the containers, we lost our old database which we inserted through application . ( This is not accepted in the real time)**

**Note: Even if we deploy latest code or if we re-create the containers, we should not loss our data . Out data should be remained in the database .**

* **If we don’t want to lose the data eve if we re-create the container then we need to make our docker container as stateful container .**

**Q) what is different between stateful and stateful ?**

**To make Docker container as stateful , we need to use docker volumes.**

**By using docker volume we able to persistent the Container data.**

* **Docker volume is used to store the data.**

**Docker volumes:**

**Applications we are executing as Docker Containers.**

**Docker containers are by default stateless**

**Once Container removed then we will lose the data that got stored in the container.**

**In real time we should not loose the data even if container got removed.**

**For Example: Database Container.**

**Application will store data in database. Even if we delete applications container or database container data should be available.**

**To make sure that data is available after the container is deleted, we will used docker volume concept.**

* **Docker volumes are used to store container data permanently**

**Volumes are the preferred mechanism for persisting data generated by and used by docker containers.**

**We have 3 types of volumes in Docker.**

* **Anonymous Volumes (without name)**
* **Named Volumes (Will have a Name) ---> Recommended**
* **Bind Mounts (storing volume on the host machine)**

**Q) what is Dangling Volume?**

**The volumes which are created but not associated with to any container are called as Dangling Volume.**

**Delete all Dangling Volume.**

**$ docker volume rm $(docker volume ls -q -f dangling=true);**

**Create Docker Volume**

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

**Display all the docker volumes**

**$ docker volume ls**

**Inspect the docker volumes**

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

**# delete docker volumes**

**$ docker volume rm <vol-name>**

**# Delete all docker volumes**

**$ docker system prune --volumes.**

**version: "3"**

**services:**

**application:**

**image: spring-boot-mysql-app**

**ports:**

**- "8080:8080"**

**networks:**

**- springboot-db-net**

**depends\_on:**

**- mysqldb**

**mysqldb:**

**image: mysql:5.7**

**networks:**

**- springboot-db-net**

**environment:**

**- MYSQL\_ROOT\_PASSWORD=root**

**- MYSQL\_DATABASE=sbms**

**volumes:**

**- app\_data:/var/lib/mysql**

**networks:**

**springboot-db-net:**

**volumes:**

**app\_data:**

**Docker Swarm:**

**Docker : it is a containerization Platform. It is used to deploy the application as containers.**

**Docker Swarm : it is an Orchestration Platform . It is used to manage the docker containers**

* **Managing Docker containers nothing but Creating/Updating/Scale up/Scale Down/Remove**

**Note: In Market we have Docker Swarm, Kubernetes, Open shift are orchestrated Platforms.**

**By using Docker Swarm Instead of running our application in one container we can manage our application at multiple containers, then easily we can scale up and scale down when go for Docker swarm concept.**

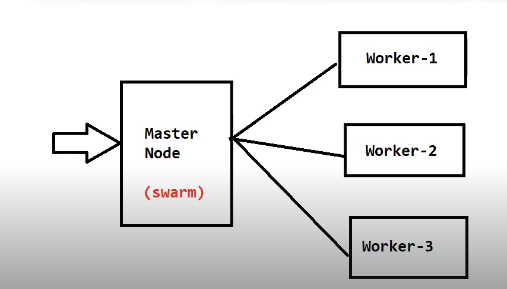
**Docker Swarm will flow Cluster Architecture to manage dockers containers**

1. **Master Node**
2. **Worker Node**

**Master Node will manage the worker node and it will assign task to worker nodes**

**Worker Node will perform the task based on master node instruction.**

* **It is a Container Orchestration Software**
* **Orchestration Means Managing Processes**
* **Docker swarm is used to setup Docker Cluster**
* **Cluster means group of servers**
* **Docker swarm is embedded in Docker Engine.(No need to install Docker Swarm Separately)**
* **We will setup Master and worker Node using Docker swarm Cluster**
* **Master Node Will schedule the tasks (containers) and manage nodes and nodes failure**
* **Worker node will perform the action (containers will run here) based on Master node instructions**



**Swarm Features:**

* **Cluster Management**
* **Decentralize Design**
* **Declarative Service Model**
* **Scaling**
* **Multi Host Network**
* **Service Discovery**
* **Load Balancing**
* **Secure by default**
* **Rolling Updates**

**Docker Swarm Cluster Setup:**

**-> Create 3 EC2 instances (ubuntu) & install docker in all 3 instances using below 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 on the master machine**

**Initialize the docker swarm Cluster.**

**$ sudo docker swarm init --advertise-addr <private-ip-of-master-node>**

**EX: $ sudo docker swarm init --advertise-addr** 172.31.81.78

**Note : if you want to act as master Node, we need to follow above commands.**

**Get Join Token From Master( this token is used by workers to join the master)**

**$ sudo docker swarm join-token worker (we can use token number using this command)**

**Ex: sudo docker swarm join --token SWMTKN-5fctpeauwtsmkgz2irwoabpz2nvwmv01r127h1ndhysy5gz9yx-8yak8aewkcir1v3tcv3ejkz27 172.31.81.78:2377**

**Note: Copy the token and execute in all worker nodes**

**If you want to one more master machine , we should execute below command**

**$ sudo docker swarm join-token master**

**Q) what is docker swarm manager quarm?**

**Ans) If we run only 2 masters then we can't get High Availability**

**Formula : (n-1)/2**

**If we take 2 servers**

**2-1/2 => 0.5 ( It can't become master )**

**3-1/2 => 1 (it can be leader when the main leader is down)**

**Note: Always use odd number for Master machines**

**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 80:80 httpd**

**Note: By default, 1 replica will be created**

**# check the services created**

**$ docker service ls**

**# we can scale docker service**

**$ docker service scale <serviceName>=<no.of.replicas>**

**Ex: $ sudo docker service scale java-web-app=3**

**# 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**

**Above command use this worker node**

**# remove docker service**

**$ sudo docker service rm <service-name>**

**docker stack deploy**