**Docker(48 note)**

**Linux**

**Manual project installation**

**Shell script**

**Ansible script**

**Terraform**

**Terraform infra + Jenkins CICD**

**Containerization**

**Physical Servers, Virtualisation, Containerization**

**Independent houses, apartments, individual rooms**

**Independent houses/Physical Servers**

**-------------------**

**advantages:**

**\* own space, privacy**

**dis advantages:**

**\* electricity**

**\* water**

**\* internet**

**\* 6months - 1year time**

**\* only one family can stay**

**\* cost is very high**

**Physical Servers --> booting time is very high**

**16GB RAM, 1TB HD**

**apartments/Virtualisation**

**-------------------**

**dis advantages:**

**\* shared area(corridor, park, lift, badminton court, pool, etc.)**

**advantages:**

**\* water, electricity, internet everything will be taken care**

**\* construction time is relatively very less**

**\* cost is medium**

**booting time is very less compared to physical server**

**a big physical server will be there --> 16GB RAM, 1TB HD, CPU**

**we will divide this big server into multiple logical servers**

**Host OS --> windows**

**VM Ware --> Hypervisor**

**Guest OS-1**

**--------**

**1GB RAM**

**Ubuntu**

**100GB HD**

**Guest OS-2**

**--------**

**1GB RAM**

**Centos**

**100GB HD**

**isolated from other servers**

**Guest OS-3**

**--------**

**1GB RAM**

**Fedora**

**100GB HD**

**0.1GB RAM, 10GB HD**

**resource utilisation is good in VM**

**single rooms**

**------------------**

**1 flat --> 3BHK**

**everyone will take one room, hall and kitchen are shared**

**advantages:**

**\* very fast**

**\* very less cost**

**we will have containers installed in VM...**

**containers are isolated from each other, system resources are shared...**

**containers take resources based on demand, it don't block resources**

**boot time is very less --> with in seconds your application is available**

**configuration == all your things**

**container/image = Fat OS(4GB/3GB) + application run time + created users + created a directory + installed application**

**AMI = server + configured the server using ansible + stop server + take AMI**

**Amazon machine image**

**docker image = Base OS(5MB-250MB) + application run time + created users + created a directory + installed application = max 500MB = immutable image = can take from DEV to PROD**

**Docker is portable**

**legacy = frontend+backend ==> war file and ear file = 100MB**

**frontend and backend divided**

**VM for monolithic apps**

**frontend = UI = VM-1 = 50 MB**

**backend = cart+catalogue+user = VM-2 = 50MB**

**Microservices --> completely independent applications**

**10/5MB you dont need VM to run a microservice. containers are a best approach**

**working in DEV, but not working in PROD**

**----------------------------------------**

**configuration changes and OS**

**docker users can only use docker, when you install docker a group called "docker" is created**

**usermod -aG docker centos**

**image vs container**

**AMI vs EC2 instance**

**docker images --> show you the images exist in server**

**docker pull nginx --> pulls the image from hub**

**docker pull nginx --> pulls the image from hub**

**nginx AMI = any OS + install nginx = nginx AMI**

**nginx image = base os + nginx installed**

**docker create <image ID>/<image-name>:version --> container will be created**

**docker ps --> running containers**

**docker ps -a --> all containers with all status**

**docker start container id --> start the container**

**docker rm container id**

**remove images --> docker rmi <image-name>/id**

**docker rmi `docker images -a -q` --> removes all images**

**pull+create+start = docker run**

**docker run -d --> to run container in background**

**docker exec -it conatainer id bash**

**0-65,535 ports**

**0-65,535 a container will have**

**Dockerfiles**

**a declarative way of creating our own image...**

**docker build -t url/username/image:version .**

**docker build image:version .**

**docker tag image:version url/username/image:version**

**docker login**

**docker push image:version url/username/image:version**

**RUN vs CMD:-**

**---------------**

**systemctl start catalogue**

**it will create one nodejs process and this process will run for infinte time...**

**CMD instruction is to make your container running..**

**RUN --> runs at the time of image building**

**CMD --> runs at the time container creation**

**every container is a process.**

**systemctl commands will not work in containers...**

**command should be foreground and attach to the screen, then you send it into background**

**100 shopping covers all are white, you know only when you open them**

**you can give some lables to them to identify easily**

**COPY vs ADD**

**--------------**

**both are used to copy the files from local to image... but ADD have 2 extra capabilities**

**1. It can directly download files from internet**

**2. It can directly untar the tar files..**

**CMD vs ENTRYPOINT**

**-----------------**

**1. CMD command can be overridden by another command at run time.**

**ping google.com ping yahoo.com**

**2. ENTRYPOINT command can't be overridden, if you try to do so, the command you are entering at run time will go and append to ENTRYPOINT**

**We can use CMD and ENTRYPOINT for best results**

**1. You can use CMD instruction to supply default arguements to ENTRYPOINT**

**2. user can always override CMD arguements from run time**

**ARG vs ENV**

**-----------------**

**ARG instruction can provide values to the Dockerfile only at buildtime. ENV variables have access in the container**

**How can I use ARG variables inside container?**

**Assign the values of ARG to ENV variables so that you can access them inside container as well**

**ARG instruction can be used before FROM instruction to provide values to FROM instruction.**

**ARG value deffnied before FROM will not be accessed after FROM**

**ONBUILD**

**-----------------**

**author of image: I want to force users of image to follow something**

**52 Note**

**docker commands**

**dockerfile instructions**

**docker compose**

**docker network create roboshop**

**once the images are ready, we can run docker compose**

**we took one OS**

**application run time, nodejs**

**user create**

**create folder**

**download application**

**systemctl**

**AMI**

**best practices**

**--------------**

**1. use official images**

**2. reduce image size by using bare minimum OS like alpine, disto, core os, etc.**

**3. use multi stage builds**

**4. use docker volumes to persist the data**

**5. use custom network to isolate containers from other projects**

**Source code --> compile --> byte code(jar) --> run byte code**

**JDK --> Java development kit**

**JRE --> Java runtime environment**

**JDK > JRE and JRE is subset of JDK**

**JDK memory > JRE memory**

**Docker Volumes**

**---------------**

**when you remove docker container, what happens to data?**

**containers ephemeral/temporary. data by default will be deleted..**

**unnamed volumes**

**named volumes**

**-p host-port:container-port**

**/opt/server --> node js file**

**roboshop:roboshop**

**addgroup**

**docker architecture**

**docker run nginx**

**1. docker shell/ docker command send a request to docker deamon**

**2. docker engine receives the request**

**3. it will check whether image is available in local or not**

**4. if available it will create container and show the response in client**

**5. if not available, it will pul from docker central hub, keep it in local.**

**6. create container and response to client**

**docker layers**

**--------------------**

**1. base image**

**creates container out of first instruction, intermediate container**

**2. runs second instruction in the container, creates image out of this**

**3. creates container out of 2 instructions.**

**4. runs the command in the intermediate container. creates image out of these 3**

**FROM node:18.19.1-alpine3.19 --> creates container-1 here**

**runs EXPOSE 8080 in container-1 --> creates image-1 out of this container-1**

**create container-2 out of image-1**

**runs this command addgroup -S roboshop && adduser -S roboshop -G roboshop in container-2 --> creates image-2 out of this conatiner-2**

**creates container-3 out of this image-2**

**RUNS**

**image --> container --> image --> container**

**RUN yum install nginx -y**

**RUN yum install git -y**

**RUN yum install gcc -y**