**DevOps- Containerization:**

**Pre- virtualization:** different os on different machine. Two applications cant run on the same port number which is costly.

**Virtualization:** different os can be installed on same machine using hypervisor concept. **Hypervisor** Is a software used to install different os on our same machine. There are 2 types of hypervisor.

1. Type 1
2. Type 2

Refer Azure notes for detailed explanation for hypervisor concept.

**Containerization or post virtualization (2013):**

In virtualization, if we are allocating 4gb of RAM and 4 CPU for one VM, there may be a chance of not using our resources efficiently when the system is idle or lack of resources when the load is more.

Time consumption is more in virtualization. Patching, updates needs to be taken care.

In containerization, instead of installing OS in different the VM’s/ containers, we will be using the base machines OS as per the requirement with the use of KERNEL. We will be installing kernel on the base machine’s OS. Kernel is the heart of OS which will command OS. So we can use the resources effectively.

**DOCKER:**

Containerization has many tools. Docker is one of them.

To install Docker on EC2 linux: **sudo yum install docker**

To check the version of docker: **docker –version**

Important document for docker:[Overview of get started | Docker Docs](https://docs.docker.com/guides/get-started/)

VM’s will be installed from the ISO which has all the binaries, libraries, kernel and etc. Containers will be created from the IMAGE which has only the basic binaries and libraries.

Once the docker is installed, we should start the docker by using the below command.

**systemctl start docker**

To create a container- **docker container run nginx** (nginx is a docker image from which a container can be created.) when we run the above command the container will be created and we can’t work on the base machine. So we can create the container in a detached mode using the below command.

If we create a container in a detached mode, the container will continue to run in the background, we will stay at our base machine. – **docker container run –d nginx**

To create a container in interactive terminal- **docker container run –it nginx bash** (bash is the shell command which provides the environment to work. So using the above command we can directly create and land inside the container and start working on it.

To come out of the container just wirte- **exit** or **ctrl+d**

To see the list of containers which is running- **docker container ls** or **docker ps**

To see all the containers available- **docker container ls –a**

To create a container with our own name**- docker container run --name mycontainer –d nginx**

**IMAGES:** Images are being pulled from the respective library.

Images are used to create the containers. It is the compressed version of ISO, which has only the basic binaries and libraries.

Images are very small compared to ISO in size, so its easy to use.

**Ex: nginx image** is a web application which can be opened in a browser. It can be used to host the website or to run any web application.

**How the docker images are being decided?**

By default, containerization will have lots of images installed in it. We can use any images as per our requirement.

All the images will be available in the artifactory. This artifactory is called as **dockerhub**. [Docker Hub](https://hub.docker.com/)

There are 3 different types images in dockerhub.

1. Docker official images- which is created and managed by docker
2. Verified publisher- approved by docker
3. Sponsored oss- open source software- not verified by docker

Image is the read layer, container is the write layer- because we cant modify the image but we can modify the container.

Docker container has the properties of inheritance, which means docker will continue to run till the assigned work or task completes. Once done, docker will go to exit mode.

To install openjdk image- **docker container –d --name testcont openjdk**

To see the list of images available in our machine- **docker image ls**

Images are ready only. Containers are read and write. We can go inside the container not inside the image.

To go inside the container- **docker container exec –it containerID or containername bash**

To use the web browser on linux**- curl –v localhost:80**

To stop the container- **docker container stop containerID or containername**

To start the container- **docker container start containerID or containername**

To delete the container- We cannot remove the running container using **docker** **container rm contID.** We can removeusing **docker** **container rm –f contID**

To change the name of the container**- docker container rename oldname newname**

To check the logs of the container**- docker logs containerID**

To kill the container- **docker kill containerID** (we can start back the killed container.)

We can copy any file or script from our local to container and vice versa.

Local to container**: docker container cp filename containername:/location ex: docker container cp useradd.sh cont1:/tmp**

Container to local**: docker container cp filelocation destinationlocation ex: docker container cp cont1: /1.txt .** (. Represents the current location)

**Pulling images from dockerhub:** Whenever are trying to create a container from any image, it will search locally if required image is available or not. If it is not there the required image will get pulled from the dockerhub by default.

**Docker image pull imagename**

If container is not having any work to do it will get exited automatically. We will get to know from the command whether our container has to do something or not

**Custom Image creation:** to create a custom image **vi Dockerfile**

Inside the dockerfile we will be writing the **instructions** line by line. These instructions are very important while creating the custom image.

**From instruction** is **FROM ubuntu**

The available readymade images will be having the set of libraries and binaries by default. So when we are creating our own custom image we can use the existing libraries and binaries from another images. To use that we will be using the **FROM** instruction. Which image can be used will be decided by the security team, higher management along with client.

**RUN instruction** is **RUN apt-get update-** to keep our system updated

The third instruction is **RUN apt-get install vim –y-** to install a software

Whenever we are creating the custom image its important to keep our file name as **Dockerfile** as the system will always search for the file named as Dockerfile.

If we want to use any other name instead of Dockerfile we need to use the below command

**docker image build –t image name –f our filename**

**Note:** we can’t create the image with the same. At the same time we cant use different names for 2 different images, as it will leads to confusion. If we are using the same name the old image will get replaced by the new even though the different base images. The base image should be of any OS.

If wanted to have the same name for different base images we can make use of the concept called as **TAGS.**

**docker image build –t myimg:dev .** (here :dev refers tag)

Inside the docker file we can write the two same instruction in one line using **&&** which is recommended.

**RUN apt-get update –y && apt-get install vim –y**

**COPY Instruction –** used to copy any file from our base machine to the container. This helps developers by copying the file every time separately.

**docker container run -it --rm --name cont1 nginx-** this command will create a container in interactive mode and as soon as we exit, the container will get deleted automatically.

**ADD instruction:** this is same as copy instruction. In addition to copying the file it will help us in untar or unzip the file.

To create tar file – **tar –cvzf tmp.tar /tmp –** creating the tar file for the directory /tmp. Means the files which is available inside the temp folder will be available inside the tar file. (c-create, v-verbos , z-zip, f-file)

USER Instruction: To change the privilege access to the users, we have another instruction called as **USER.** Before writing the USER instruction we need to add the users by using RUN command.

Command to create an user**- useradd testuser**

If any user wants/ needs to be test and root user we can use the below command. **docker container run –it --user root** (the username which is already available) **–name mycon1 myimg:2**

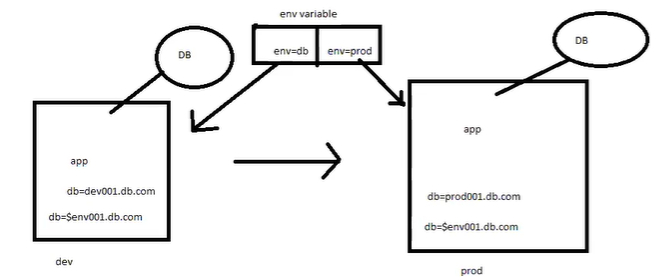
**WORKDIR Instruction**: Working Directory change Instruction:

Whenever we are creating a container, we are directly landing at /path. If we want to change the landing path we can use the below instruction.

**LABEL Instruction:**

Used to mention some information about the image or the team or the person who developed the image. Like label on the gift box mentioning the names.

**Environment Variable:** is used to remove the hardcode values from our code. Without making any changes in the code we can use in different environments like dev, prod, pre-prod and etc.



**Environment Instruction**: declaration:  **ENV app= dev** (this environment variable should be in the form of key, pair value. Here the **key** is **app** and the **pair** is **dev**). To check the value of the environment variable we use **echo $variablename or key.**

We can change the environment directly without creating going inside the docker.

**docker container run –it --env app=test --name mycont myimg**

**CMD Instruction:** used to give or mention the tasks which out container needs to perform.

**CMD [“sleep”,”50”]**

Watch command used to do the auto refresh our linux command**: watch docker ls**

To see the content of an official docker image is **docker image history nginx.**

**FROM** ubuntu

**RUN** apt-get update-y

**RUN** apt-get install nginx –y

**LABEL** author=devopsteam

**ENV** app=dev

**RUN** mkdir /app

**RUN** useradd testuser

**USER** testuser

**WORKDIR** /app

**COPY** myfile /app

**ADD** tmp.tar /app

**CMD** ["nginx", "-g", "daemon off;"]

**ENTRYPOINT** [“bash”, “useradd.sh”]

**EXPOSE** 80

If we have two CMD instruction inside the dockerfile, the latest CMD will get executed, first one will be overridden by the second one.

**ENTRYPOINT Instruction:**

It is same as the CMD instruction, but Entrypoint will accept the runtime arguments as well.

**Real time example: adding multiple users.**

Create a file userfile.sh- copy to dockerfile- build an image and create the container. So we can add multiple users using the same image. For this we can make use of ENTRYPOINT instruction.

**ENTRYPOINT [“bash”, “useradd.sh”]**

**EXPOSE Instruction:** it will help us to mention the port number on which our application should run. We can get the details from the developer.

**EXPOSE 80** (80 is the nginx port number)

**Changing the storage location of our container and images:**

By default all the images, containers will be available in **/var/lib/docker.** We can change the location where we can store our images and containers, to save some space or to avoid any crashes.

Go to cd /etc/docker – then create vi daemon.json - inside that { "data-root": "/mnt/storage" } - create the storage directory if it is not present already- after that try to create a container and check whether the newly created container is getting saved inside the new location. If not, we can restart the docker by following the below steps.

1. Systemctl stop docker
2. Systemctl restart docker
3. Systemctl daemon-reload
4. Systemctl restart docker

**Dangling Images:** Whenever we are updating or overriding the Dockerfile and creating image with same name the 1st /previous image will become **none**. This **none** image is called as dangling image.

**Cleanup of Docker:** we can do the cleanup by manually deleting the containers, images. Another way to cleanup is to use the below command. It will help us in removing

* All stopped containers
* All networks not used by atleast one container
* All dangling images
* All dangling build cache

The command is: **docker system purne** (deletes only the things which are not in use)

Docker hub pulls the docker images from the library only. As all the images are being stored inside the library.

We can push our custom images to the dockerhub under our account only. We cant put under the libraries as we will not have access for those. Only verified images will be available inside the libraries.

To push the images under our account: **docker image tag imagename accountname/imagename** + **docker image push accountname/imagename-** before this we need to do **docker login** by entering username and password.

When there is situation to share our container between the team, we cant move our container from one place to another but we can move our images. So using the existing container we can create an image and that image can be shared with others after pushing into Dockerhub.

Below is the command to create the image from/out of the container itself.

**docker container commit containerID imagename**

There is one another way to share our image is convert the image into tar file.

**Docker image save imagename > Imagename.tar** – this command will convert our image into .tar file. So that we can share the .tar file directly to other team members without moving into dockerhub.

*When we are sharing the tar file, there is a chance that our file will get corrupted.*

Untarring the image- **docker image load <img.tar**

To check our application is running inside the container or not, we use the command-**curl localhost.** If its working fine inside the container then, there may be some other issue.