**Docker:**

* Open-Source platform that enables the developers to automate deployment, scaling and management of applications using containers.
* Containers allow applications to run consistently across different environments, eliminating issues related to dependency management.
* **Key Features of Docker:**
  + **Containerization:** packages applications along with dependencies, libraries and configuration files.
  + **Portability:** containers run on any system that has Docker installed, making development and deployment seamless.
  + **Scalability:** containers can be easily replicated and scaled up or down.
  + **Light Weight:** Containers share the host OS kernel, making them more efficient than virtual machines.
  + **Rapid Deployment:** Fastener application deployment and testing.
* Docker is designed to create, run and deploy applications.
* Docker is an Containerisation tool.
* Docker can be done by Virtualization/CLI.
* Virtualization: can be done by using VMware, by using this multiple Operating Systems can be managed.
* An example of VMware is Hypervisor.
* Can even manage Heavy Weight that is Hardware Components(CPU,Memory,Hard Disk and Utilization).
* Effective Utilization of Hardware resources → Dockers.

**Containerisation:**

* Containers are nothing but Operating Systems.
* Container is Server.
* Containers are Lightweight to move/ carry the application → that means no need to take hardware resources are required.
* Less utilization of Hardware resources are done.

**What is Docker?**

* Containers allow developers to package up an application with all of the parts it needs, such as libraries and other dependencies and ship it as one package.
* Dockers are very light weight.
* On a single host we can run multiple containers by using host port IP.

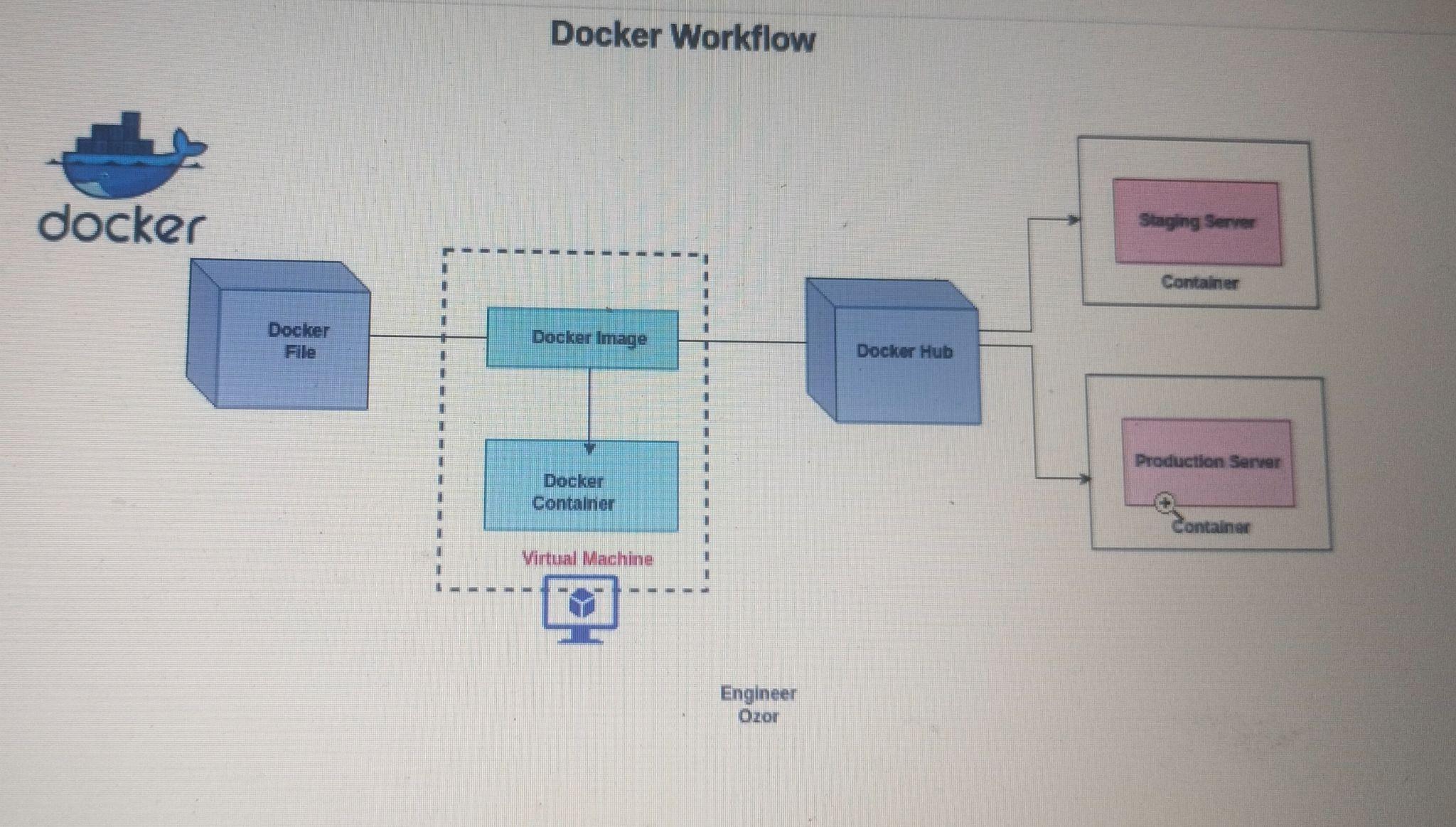
**Why Dockers?**

* In Order to handle our application safely to the end user.
* Official Repository for Dockers: hub.docker.com

**Docker file vs Docker Image vs Docker Container:**

| Docker File | Docker Image | Docker Container |
| --- | --- | --- |
| A Docker file is set of Instructions to build a Docker Image | A packaged version of the application created from a Docker file | A running instance of a Docker image |
| Acts a Blueprint for building Docker Image | Acts as a template to create a containers | Acts as an executable application environment |
| Used to define the environment, dependencies, and configurations needed for an application. | Contains all the dependencies, libraries and files required for a container. | Uses the file system and environment from an image to execute an application. |
| Written in a simple scripting format with instructions like **FROM, RUN, COME, CMD** etc. | Built using **docker -t <image\_name>**  From a docker file. | Created and run using **docker run <image\_name>** |
| Does not run itself, defines how an image should be built | Cannot execute but can be used to start containers | Uses computing resources (CPU, RAM, storage) when running. |
| Must be written manually by the developer. | Can be shared on Docker Hub for others to use. | Cannot be shared directly but can be replicated using images**.** |

**Workflow:**



* **Docker file:**
  + A text file with a set of instructions to build a docker image.
  + An automation of Docker Image.
  + Should be saved with Capital-D.
* **Docker Image:**
  + The Dockerfile is built into a Docker Image.
  + The image acts as a read-only template that contains everything needed to run the application.
  + This image is then stored locally or pushed to Docker Hub.
* **Docker Container:**
  + An instance of running Docker.
  + Runs on Virtual Machine/ Physical Server.
* **Docker Hub:**
  + Docker Image can be uploaded to Docker Hub.
  + This allows developers to share the image with teams or deploy it in different environments.
* **Deployment to Staging and Production Servers:** 
  + The Docker Image is pulled from Docker Hub and deployed as a container

**Docker Client - Server Architecture:**

* Docker contains Client-Server Architecture.
* One client and Many Servers.
* **Docker Client:**
  + That is the part of the Docker you, as a user interact with.
  + It is a Command Line Tool(or API) that sends the instructions to the Docker Server.
* **Docker Daemon (Sever):**
  + Daemon is the core engine of Docker that handles all container-related operations.
  + It listens for all the requests from Docker client and performs tasks such as building, running, and managing containers/images.
  + Also interacts with Storage and Networking to ensure containers function smoothly.
* **Rest API:**
  + The communication between the Docker Client and Docker Daemon happens via RESTful API.
  + This API provides a standard way for the client to send commands and receive the responses from the Daemon.
* **Docker Engine:**
  + Includes Docker Daemon, Docker CLI and REST API.
  + Provides the full functionality to build, ship, and run containers.
* **Docker Images:**
  + Pre packaged applications that contains all the dependencies to run on OS.
  + Stored in Local storage and Remote repositories.
* **Docker Containers:**
  + Running instances of Docker images, isolated from the host system.
  + Containers use namespaces for resource isolation.
* **Docker Registry:**
  + Stores and distributes the Docker images.
  + Developers can pull or push images.
* **Storage and Networking:**
  + **Volumes:** Persist storage for containers.
  + **Networks:** Allows communication between containers.

**Docker File Instructions:**

* We need to write a docker file first.
* Docker file name should be unique (first letter should be capital)
* To write a docker file we should know the set of instructions
* The Docker file should start with the Capital Letter.
* Usually present in the Project root directory.
* **FROM:** used to specify the base image(from where we are pulling the image).
* **LABEL:** labels are like meta info of the image(detail info of the image).
* **RUN:** used to run the commands on docker images.
* **WORKDIR:** it makes the directory as the current working directory. If the folder is not available, then it will automatically create the folder.
* **EXPOSE:** to open a port of the container (maps the container to a port).
* **CMD:** used to start the application.
* **ENTRY POINT:** similar to CMD command.
* **COPY:** to copy the file from local host to docker image and source can be only on local file system.
* **ADD:** adding a file to the docker image and source can be local file and remote URL.

**Creating the Docker Hub Account:**

* Go to Chrome → search → Docker Hub → Signup page → Create account and login with your credentials → you will get a home screen after logging into account successfully.

**Docker Commands:**

* Docker –version : to know the version of the Docker
* ifconfig -a: to check the configuration
* docker -o: it is a bridge network
* docker info: to see the complete info of docker
* docker image: to seed the no.of docker images present
* docker images ls: same as docker images command
* docker ps: to see no.of containers
* docker ps -a: shows running/ stopped containers.
* docker pull imagename: server same like apache tomcat, pulled from official website.
* To check whether images pulled/not:
  + docker image: contains image id
  + docker inspect imageId: it will display the pulled image
* Now, we got the image ,next we need to create container:
  + docker run -itd -name -cid
* Container got created, need to display the the no.of containers:
  + docker ps: to see the no.of containers
* Note:
  + If I delete the docker container, then I can get the backup.
  + We can create ‘n’ no.of containers for single docker image
  + If we delete the docker image then I can’t have any backup
  + Docker login: username, password
  + Inorder to take backup,we need to create tag first, then we need to push
  + Command to create a tag is: docker tag <image name>: latest docker hub account
  + <Username/Dev: Jenkins>