**DOCKER**

**What is Docker?**

|  |  |
| --- | --- |
|  | Docker is a containerization platform which packages application and all its dependencies such that binaries, libraries, utilities, etc. together the form of Container, to ensure that your application works seamlessly in any environment be it Development/ Test/ Production.  Containers uses the underlying Kernel of host OS itself.  Resource usage can be defined w.r.t to the container like CPU, RAM utilization.  Multiple containers can be created using same Application.  Docker image is the package of application and the dependencies within itself. |

**What is Docker Container?**

Docker container is an isolated application platform which contains everything needed to run the application. They are built from one base docker image and dependencies are installed on top of the images as “image layers”.

**Benefits of Containers:**

* Containers are lightweight; can be easily shared via Docker Hub.
* Easily run applications by packaging them into containers.
* Containers consumed by container is less than the RAM consumed by VMs.

**What is Docker Image?**

Docker image is collection of various libraries, binaries, dependencies along with your application. An image can contain multiple image layers. Container can be executed by running image.

|  |
| --- |
|  |

**Some basic** **Commands:**

* **docker images** (to see images that exist in the system)
* **docker pull <image-name>:<version-number>** (it checks at docker hub and pulls the latest images to the system)
* **docker run <image-name>** (it runs container based on the image mentioned)
* **docker stop <container-name/id>** (to stop container)
* **docker ps** (gives the list of actual containers running)
* **docker ps -a** (to see the containers that were running previously and running at present)
* **docker run -d <image-name>** (running an image in detached mode)
* **docker run -p<port-of-host>:<container-port> <image-name>**
* **docker container prune** (to delete all the containers)
* **docker run -d -p <docker-port>:<application-port> --name <container-name-you-want> <image-name>**
* **docker build –no-cache -t <image-name> .** (not to consider cached layers)

**Some important commands explained:**

* **docker run –-name nginxdemo -P -itd nginx**
* **docker run**: Command to run docker container.
* **--name nginx**: Assigns the name **nginxdemo** to the running container.
* **-P**: Publishes all exposed ports from the container to random ports on the host system
* **-itd**: Start the container in interactive mode (**-it**) with a pseudo-TTY allocated and detaches (**-d)** the container to run in background.
* **nginx**: Specifies the Docker image to use for creating and running the container.

**Images:**

nginx – web server image used for reverse proxies.

**How to create image of your application?**

* Create a file named **Dockerfile** inside your application folder.
* Write instruction in Dockerfile i.e., dependencies, images, port, commands, etc.
* Create image of your application through the command:
  + **docker build -t <application-name>:<version(1.0)> .**

**Debugging a Container:**

* **docker logs <container-id/name>** (to see the logs of the container)
* **docker run –name <name-to-container> <image-name>** (creates and runs new container with mentioned name)
* **docker exec -it <container-id/name> /bin/bash** (gives command line for the container to execute command on the container) and **exit** (to exit from the container terminal)

**Docker Network:**

* Docker creates its isolated docker network where the containers are running in.
* When two containers are deployed in the same docker network, they can communicate to each other using just container names.
* The application that run outside the docker network is going to connect to the containers on docker network from the host using **localhost:<port-number>**
* Command> **docker network ls** - gives the list of docker networks.
* Command> **docker network create <network-name>** - helps to create a user required docker network.

**Note: Commands to provide privileges to databases**CREATE USER 'your\_username'@'host.docker.internal' IDENTIFIED BY 'your\_password';

GRANT ALL PRIVILEGES ON your\_database.\* TO 'your\_username'@'host.docker.internal';

FLUSH PRIVILEGES;

**DOCKER COMPOSE**

**Introduction:**

* Docker compose helps to start multiple containers with a single command.
* To achieve it, we need to have **compose.yaml** file.
* We can run this application with the command [**docker compose up -d**] in your project directory, this command builds and runs all the services listed in the compose file.
* When developing with the docker, if we want to automatically update and preview running services as you edit and save your code. Then use command [**docker compose watch**].
* You can delete and restart everything.
* If there are two images and running application fails, then try to run depended image first and frontend next.

**Some important commands:**

* docker-compose up
* docker-compose up –build
* docker-compose down
* docker-compose down –rmi all && docker-compose up --build