

**Problem** – When we are developing the software applications we develop it in the dev environment once the development is done move to the testing environment. Sometimes whatever the code is working in dev it may not work in QA to Prod due to dependencies or used versions or OS or, other compatibilities.



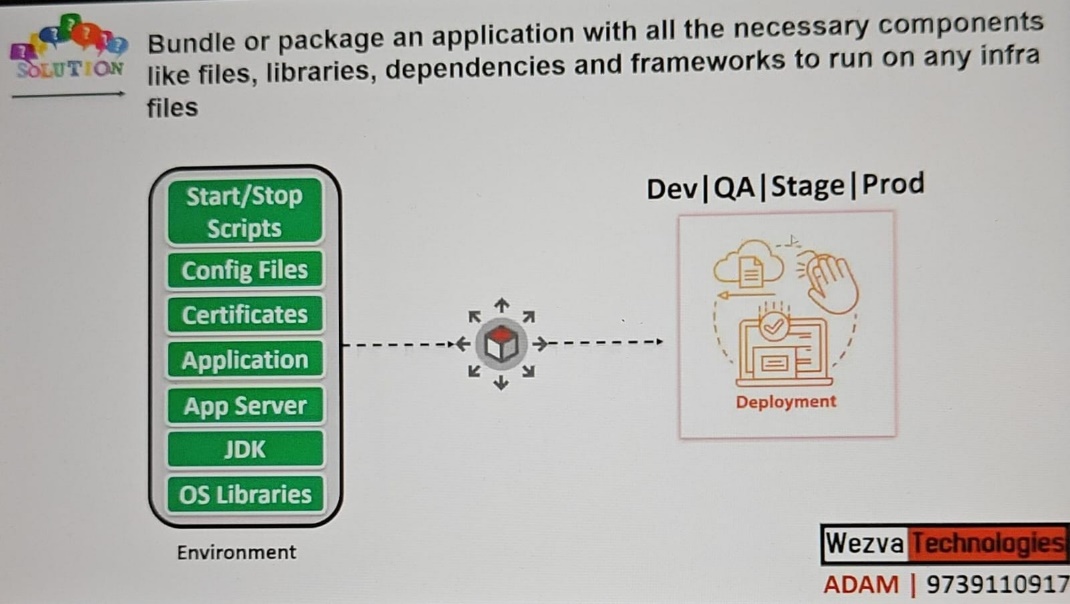
**Solution**

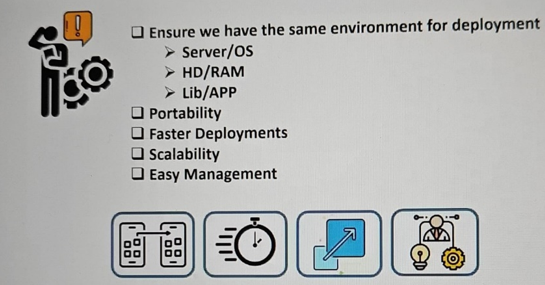
[Docker](https://www.geeksforgeeks.org/introduction-to-docker/)is an open-source [containerization](https://www.geeksforgeeks.org/containerization-using-docker/)platform by which you can pack your application and all its dependencies into a standardized unit called a container.

Containers are lightweight which makes them portable, and they are isolated from the underlying infrastructure and from each other container.

Docker is an open-source platform that enables developers to build, deploy, run, update, and manage containers.

Containers are standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.



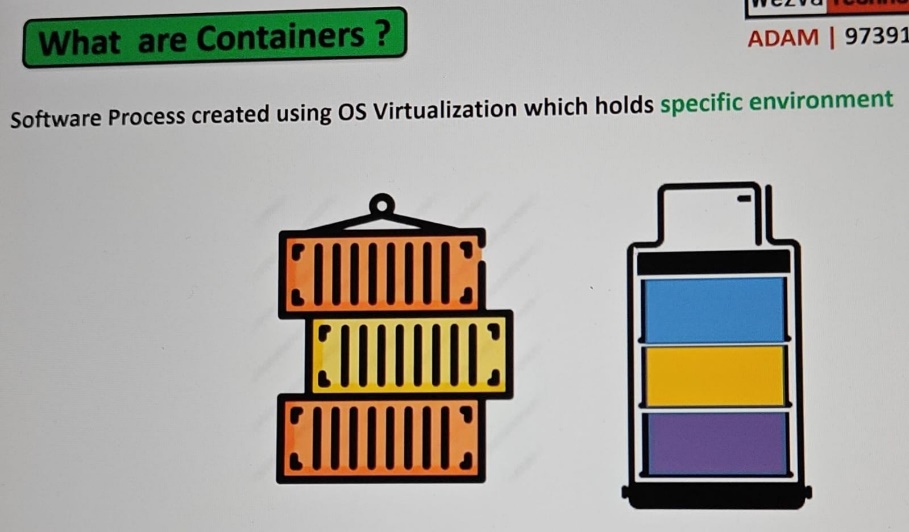


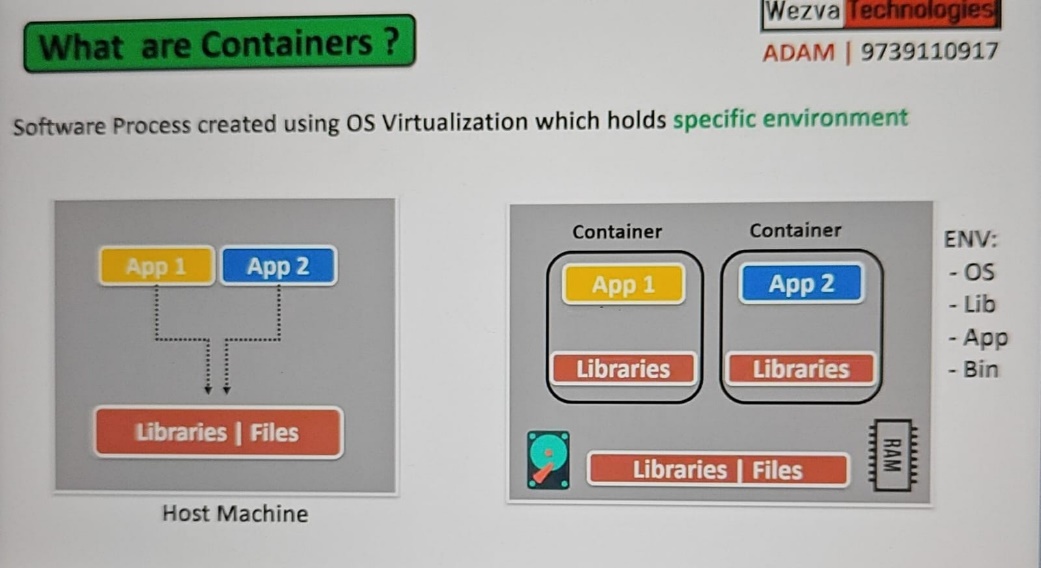
**Why Docker is popular?**

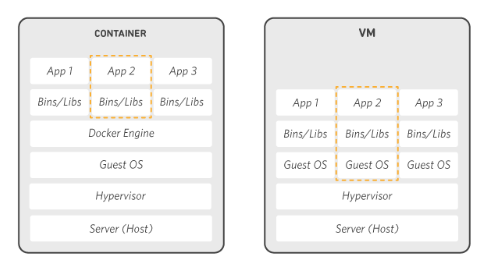
Docker gained popularity due to its impact on software development and deployment.

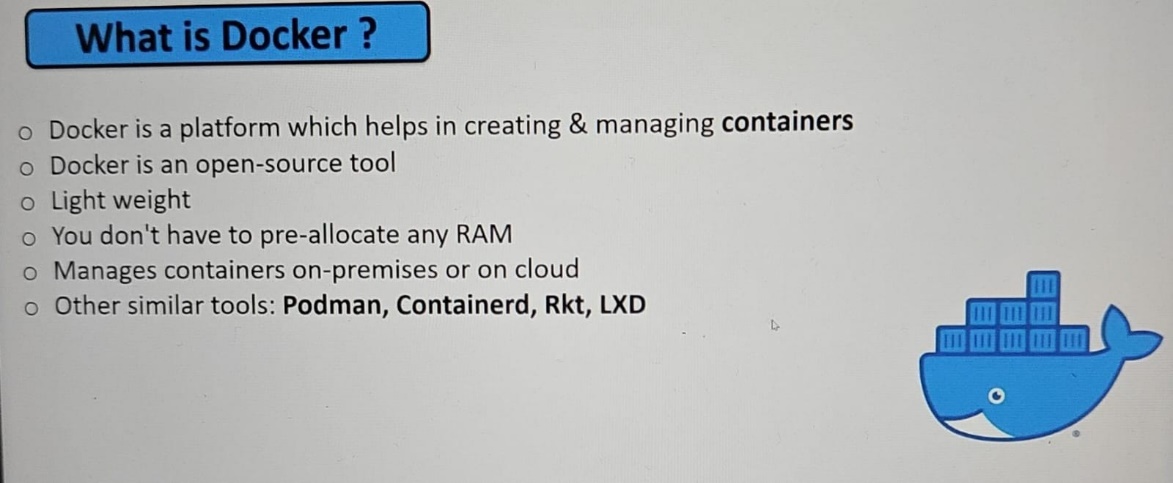
The following are some of the main reasons for Docker becoming popular:

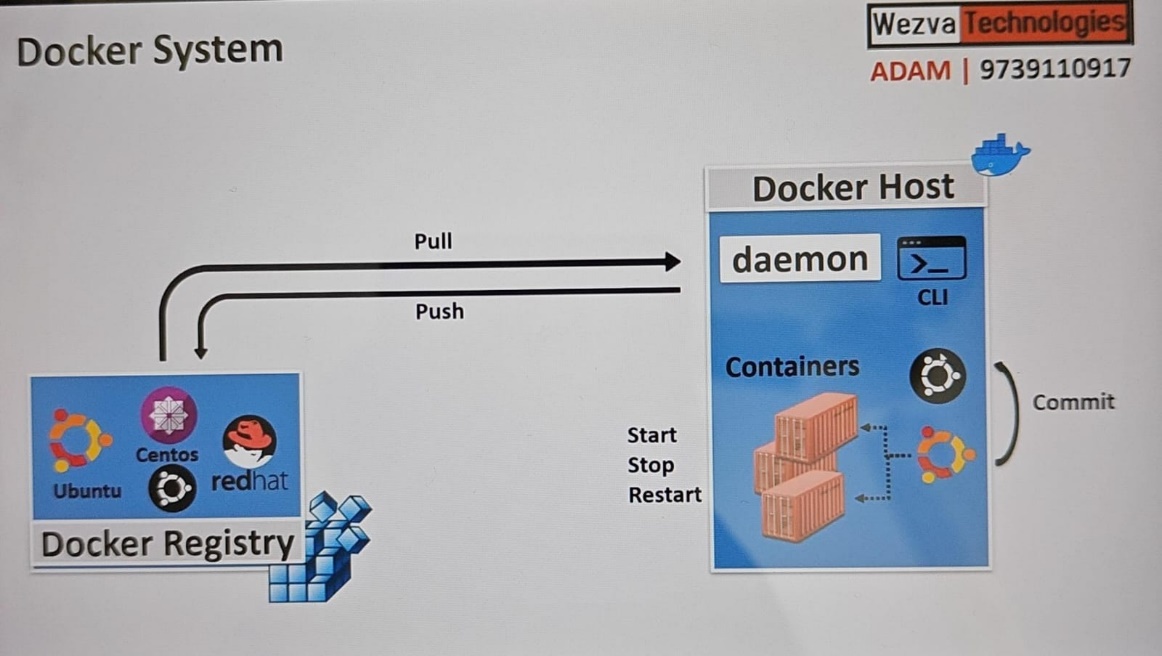
1. **Portability:**Docker facilitates the developers in packaging their applications with all dependencies into a single lightweight container. It facilitates ensuring consistent performance across different computing environments.
2. **Reproducibility:**Encapsulating the applications with their dependencies within a container ensures that software setups remain consistent across the development, testing, and production environments.
3. **Efficiency:**Docker through its container-based architecture optimizes resource utilization. It allows the developers to run multiple isolated applications on a single host system.
4. **Scalability:**Docker’s scalability features facilitated the developers in making easier of their applications handling at time of workloads increment.

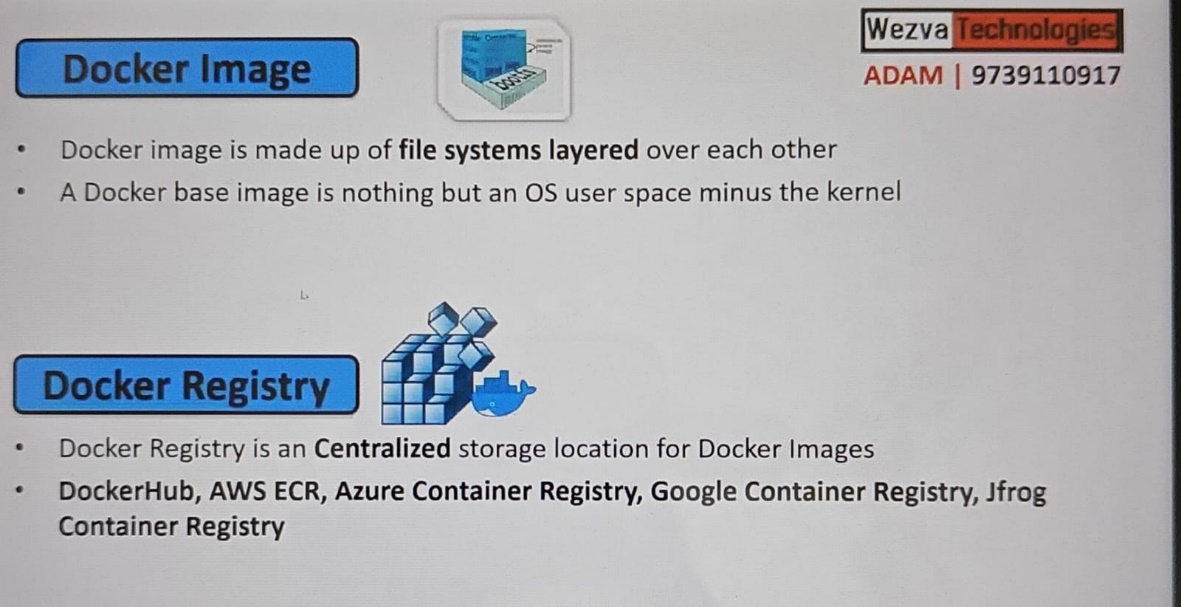












When you install a docker in any machine we call it a Docker Host. When you install a docker you get two software’s called CLI [Docker] and Daemon [Docker engine].

Docker Daemon – It is a background process and it always running. As users, we cannot communicate directly. We use CLI to communicate it.

Docker uses a client/server architecture. The following is a breakdown of the core components associated with Docker, along with other Docker terms and tools.

**Docker host**: A Docker host is a physical or virtual machine running Linux (or another Docker-Engine compatible OS).

**Docker Engine:** Docker engine is a client/server application consisting of the Docker daemon, a Docker API that interacts with the daemon, and a command-line interface (CLI) that talks to the daemon.

**Docker daemon:**Docker daemon is a service that creates and manages Docker images, by using the commands from the client. Essentially the Docker daemon serves as the control center for Docker implementation.

**Docker client:**The Docker client provides the CLI that accesses the Docker API (a [REST API](https://www.ibm.com/topics/rest-apis)) to communicate with the Docker daemon over Unix sockets or a network interface. The client can be connected to a daemon remotely, or a developer can run the daemon and client on the same computer system.

**Docker objects:**Docker objects are components of a Docker deployment that help package and distribute applications. They include images, containers, networks, volumes, plug-ins and more.  
  
**Docker containers:**Docker containers are the live, running instances of Docker images. While Docker images are read-only files, containers are live, ephemeral, executable content. Users can interact with them, and administrators can adjust their settings and conditions by using Docker commands.

**Docker images:**Docker images contain executable application source code and all the tools, libraries and dependencies the application code needs to run as a container. When a developer runs the Docker image, it becomes one instance (or multiple instances) of the container.

Building Docker images from scratch is possible, but most developers pull them down from common repositories. Developers can create multiple Docker images from a single base image and will share their stack's commonalities.

Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to an image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be reused in other projects.

Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—like adding or deleting files—are saved to the container layer, and these changes only exist while the container is running.

This iterative image-creation process increases overall efficiency since multiple live container instances can run from a single base image. When they do so, they use a common stack.

**Docker build**: Docker build is a command that has tools and features for building Docker images.

**Docker file:**Every Docker container starts with a simple text file containing instructions for how to build the Docker container image. Docker file automates the process of creating Docker images. It's essentially a list of CLI instructions that Docker Engine will run to assemble the image. The list of Docker commands is vast but standardized: Docker operations work the same regardless of contents, infrastructure, or other environment variables.

**Docker documentation**: Docker documentation, or Docker docs, refers to the official Docker library of resources, manuals and guides for building containerized applications.

**Docker Hub**: Docker Hub6 is the public repository of Docker images, calling itself the world's largest library and community for container images7. It holds over 100,000 container images sourced from commercial software vendors, open-source projects and individual developers. Docker Hub includes images produced by Docker, Inc., certified images belonging to the Docker Trusted Registry and thousands of other images.

All Docker Hub users can share their images at will. They can also download predefined base images from the Docker filesystem as a starting point for any containerization project.

Other image repositories exist, including GitHub8. GitHub is a repository hosting service well known for application development tools and as a platform that fosters collaboration and communication. Users of Docker Hub can create a repository (repo) that can hold many images. The repository can be public or private and linked to GitHub or Bitbucket accounts.

**Docker Desktop:**Docker Desktop is an application for Mac or Windows that includes Docker Engine, Docker CLI client, Docker Compose, Kubernetes, and others. It also provides access to Docker Hub.

**Docker registry:**A Docker registry isa scalable, [open-source](https://www.ibm.com/topics/open-source) storage and distribution system for Docker images. It enables developers to track image versions in repositories by using tagging for identification. This tracking and identification are accomplished by using Git, a version control tool.

**Docker plug-ins:**Developers use plug-ins to make Docker Engine even more functional. Several Docker plugins supporting authorization, volume and network are included in the Docker Engine plug-in system; third-party plug-ins can be loaded as well.

**Docker extensions:** Docker extensions enable developers to use third-party tools within Docker Desktop to extend its functions. Extensions for developer tools include Kubernetes app development, security, observability and more.

**Docker Compose:**Developers can use Docker Compose to manage multicontainer applications, where all containers run on the same Docker host. Docker Compose creates a YAML (.YML) file that specifies which services are included in the application and can deploy and run containers with a single command. Because YAML syntax is language-agnostic, YAML files can be used in programs written in Java, Python, Ruby and many other languages.

Developers can also use Docker Compose to define persistent volumes for storage, specify base nodes and document and configure service dependencies.

Containerization

\* light weight

\* doesnt have kernel

\* scalable

**Docker**

\* Image Management

\* Container Management

**Install Docker**

$ sudo su -

$ apt update

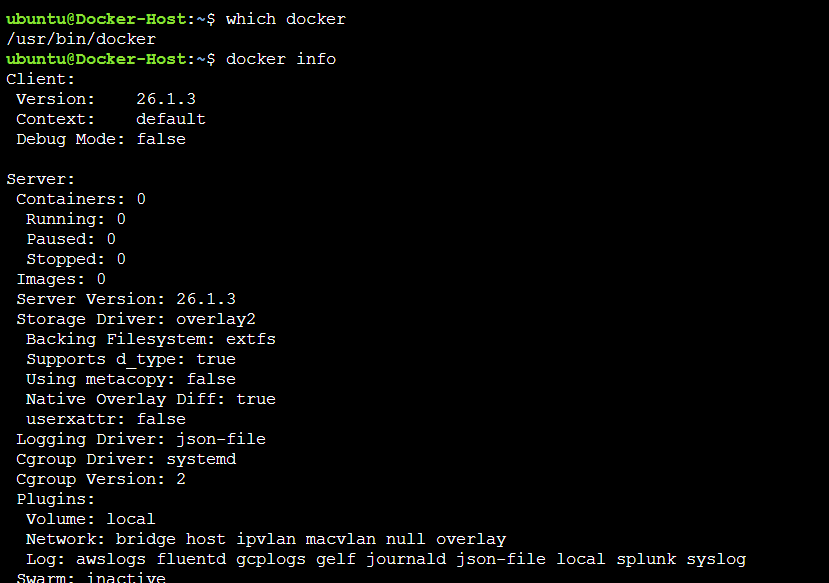
$ apt install -y docker.io

$ systemctl status docker

$ systemctl start docker

$ docker info

$ sudo usermod -a -G docker **ubuntu**



$ docker run --name <Cname> -it|-d -p <HP>:<CP> -v <HD>:<CD> <Image> <StartupCMD>

- Download the Image from the registry

- Create a new container, unique ID

- Start the container, execute the startup cmd

- Attach to the container interactively

**-it** [Interactively attached to the terminal] –

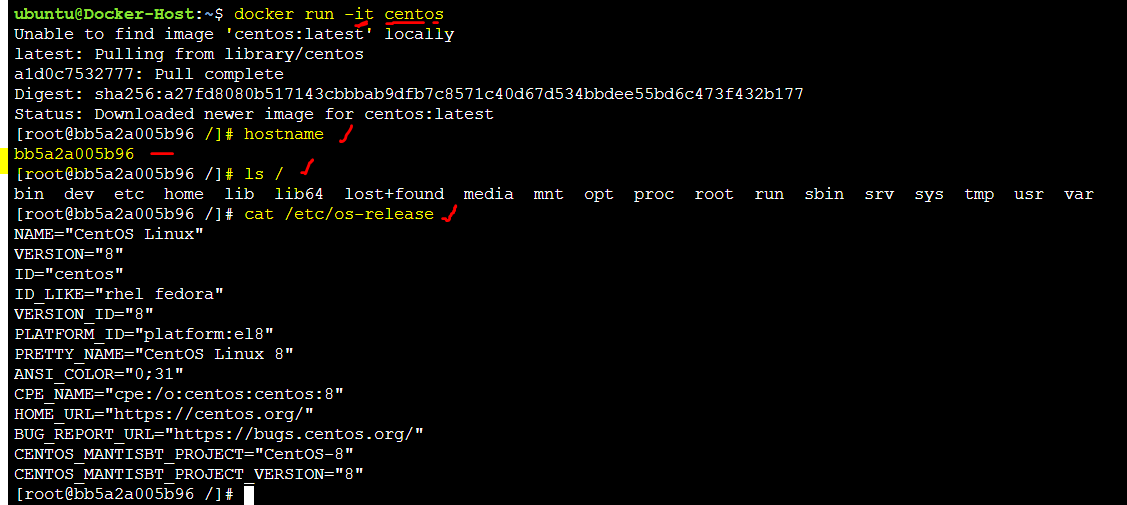
You will be in the container not on the Host.

-it is used for beginning to learn not in real-time use.

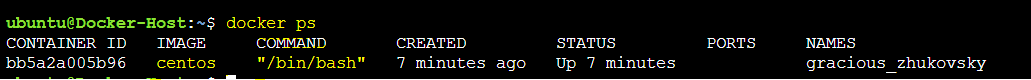
It downloads an image, creates a container, and attaches it to the shell

**-d** [Interactively detached mode to the terminal]

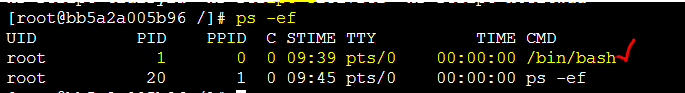
**-p** [Port] - It contains the Host Post and Container port



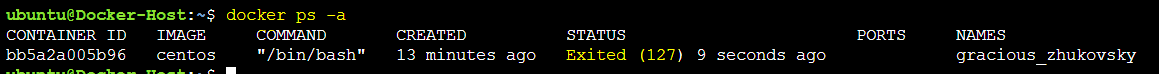
-Host



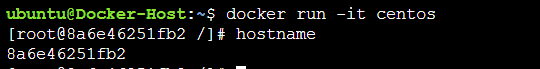
* Container

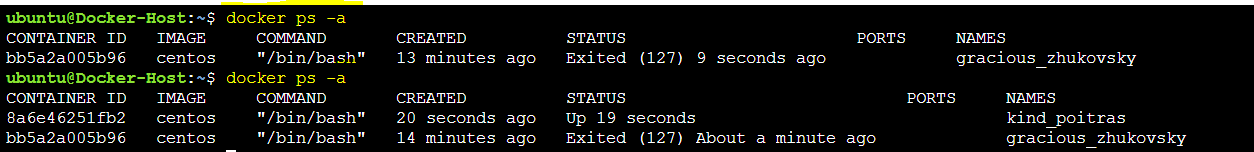


Come out from the container then check in the host it no more existed.



If you create again it will create a new container [ You can refer the ID ]





**Note:** The startup command is /bin/bash by Default but you can give diff shell

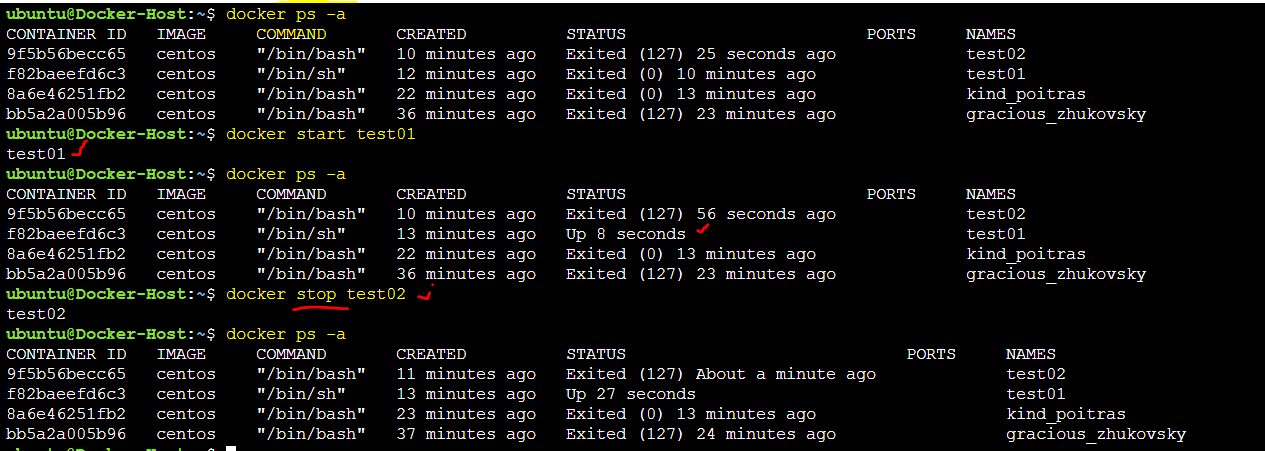


The container and Host will have a different file system

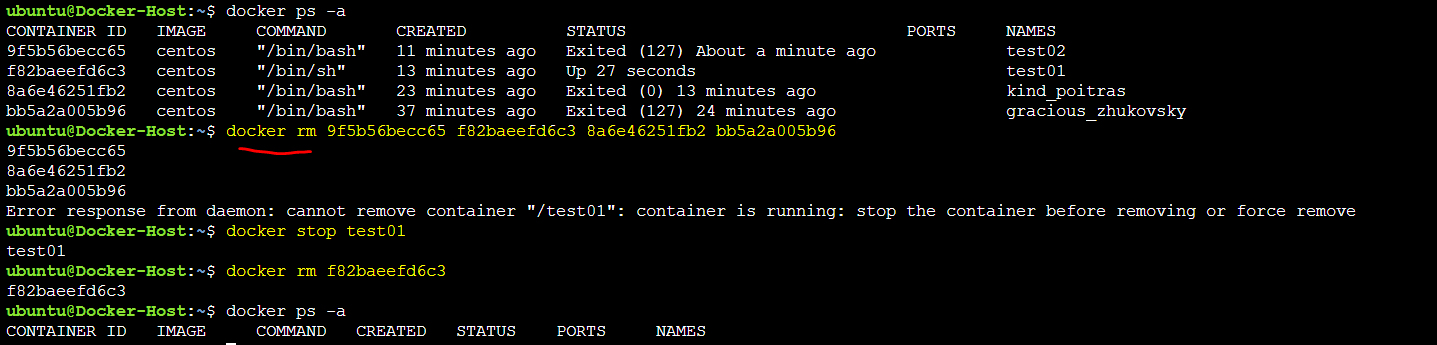
The container and the Host can have different OS or the same.

Every container has its ID or Name.

**IMP** - The containers are epiformal or use and throw. We are never going to use the container as soon as stopped. Instead, we throw it away and create a new one. We have images so we can create as many as we can. For the Interview purpose, we can say any container can stop and start but in real time we create a new one. Very specific/rare case we do stop and start.



If you want to remove/delete the container

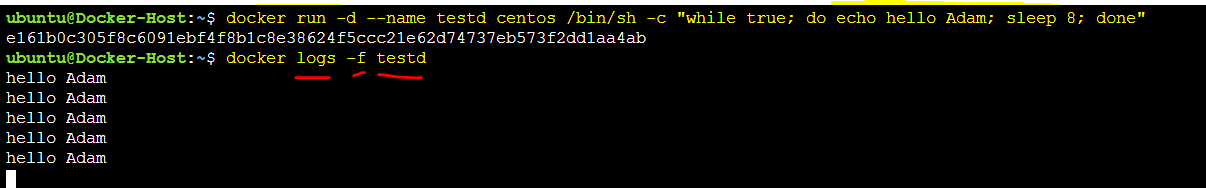


In the above case as soon as we come out from the container the container will also be stopped. Just assume if your application is deployed then your application also comes out.

To avoid that we have an option called detached mode which will help to run it in the background mode.

If you are running in the background mode we check what is happening in the container logs.

$ docker run -d --name testd centos /bin/sh -c "while true; do echo hello Adam; sleep 8; done" $ docker logs -f testd



$ docker images

$ docker ps -a

$ docker start <CID|Cname>

$ docker stop <CID|Cname>

$ docker attach <CID|Cname>

$ docker rm <CID|Cname>

$ docker logs -f <CID|Cname>

$ docker exec <CID|Cname> <cmd>

$ docker run -it centos

$ docker run --name test00 -it centos

$ docker run --name test01 -it centos /bin/sh

$ docker run -d --name testd centos /bin/sh -c "while true; do echo hello Adam; sleep 8; done"

$ docker exec testd ps -ef

$ docker exec -it testd /bin/bash

$ docker run -it --rm centos /bin/bash

$ docker run --rm --name myapache -p 80:80 -d httpd

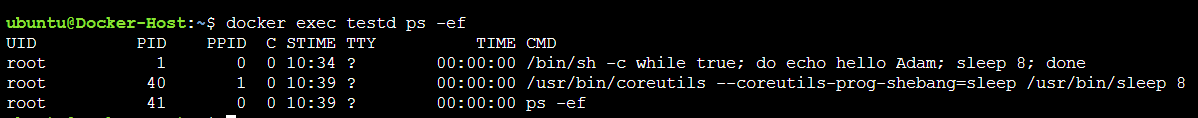
$ docker run --rm --name mynginx -p 8081:80 -d nginx

$ docker run --rm --name myjenkins -p 8080:8080 -d jenkins/jenkins

$ docker run --name c1 -it -v /tmp/host:/tmp/cont centos /bin/bash

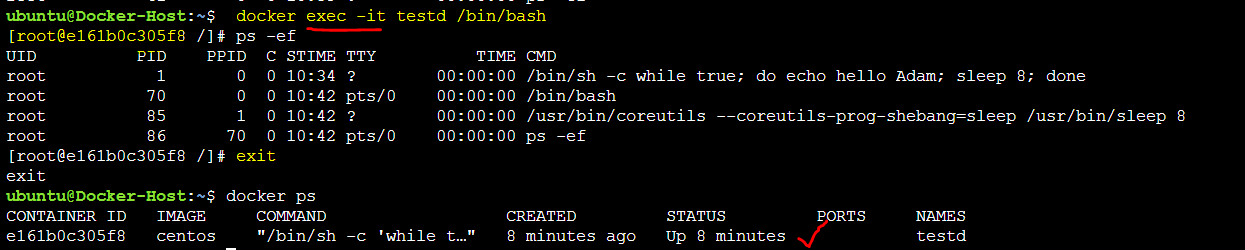
If want to execute something in the container we use EXEC. It will run the command without attaching the container.

$ docker exec testd ps -ef



If you want to go to the container and do something and exist without stopping the container

$ docker exec -it testd /bin/bash

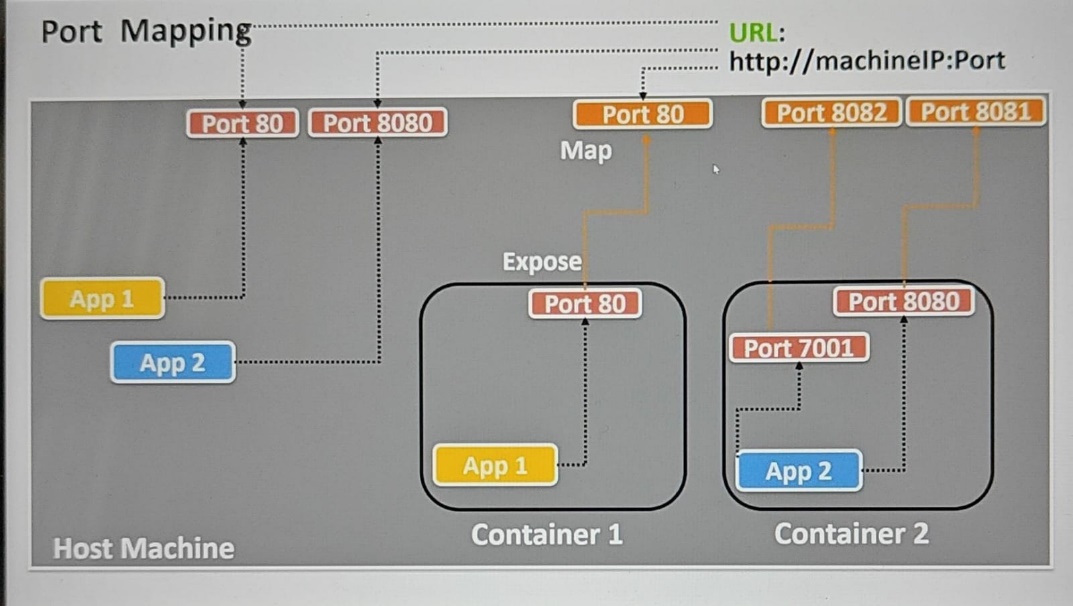


Port Mapping

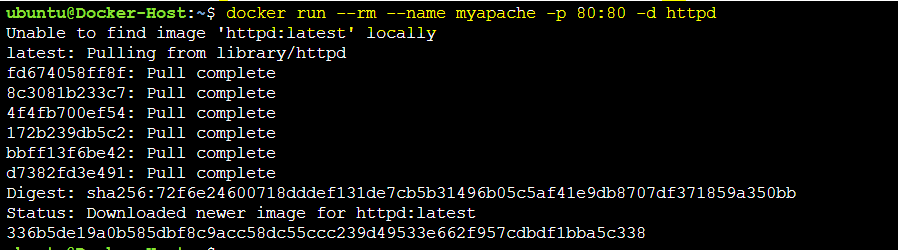
Look for the available port in the host machine. You can’t change the container port.

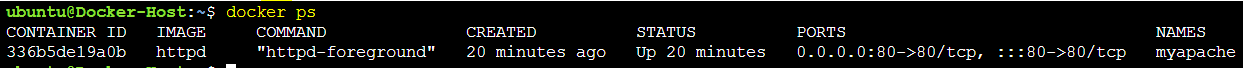
There are some cases one container can have multiple ports. Like, One port is for web applications like Java, One port is for debugging, One port is for Management, and one port is for doing monitoring.

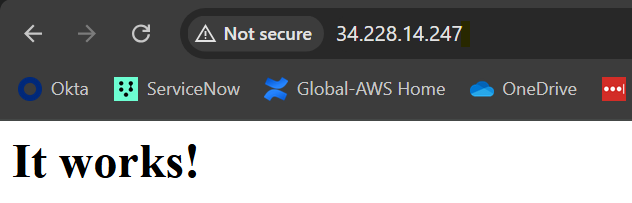
Port mapping can only be possible while creating the container. Once the container is created port mapping can’t be done and if you want to map then need to delete then create a new one.



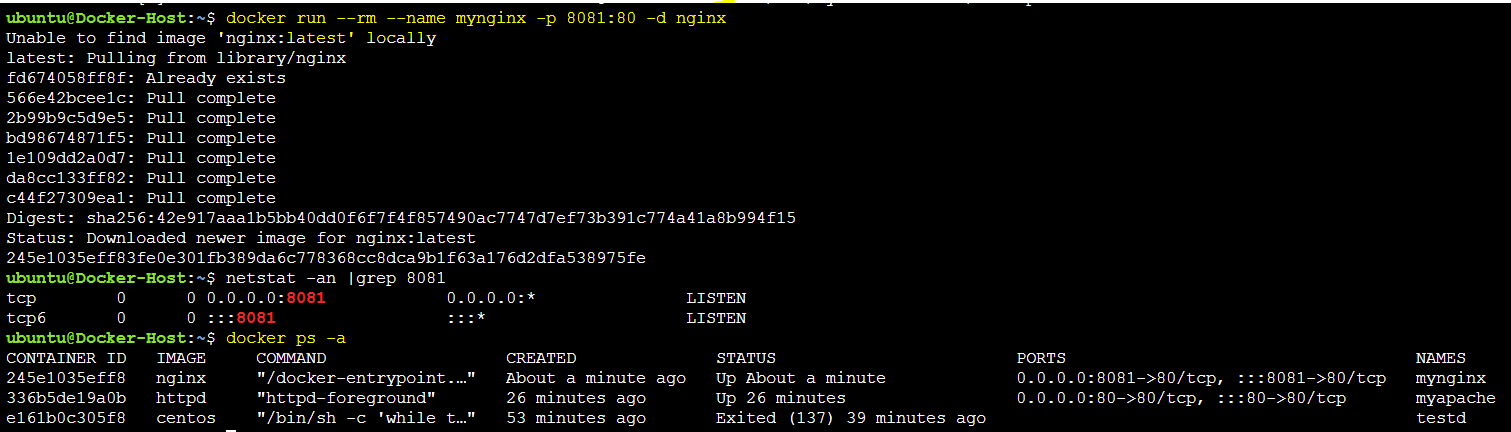
$ docker run --rm --name myapache -p 80:80 -d httpd

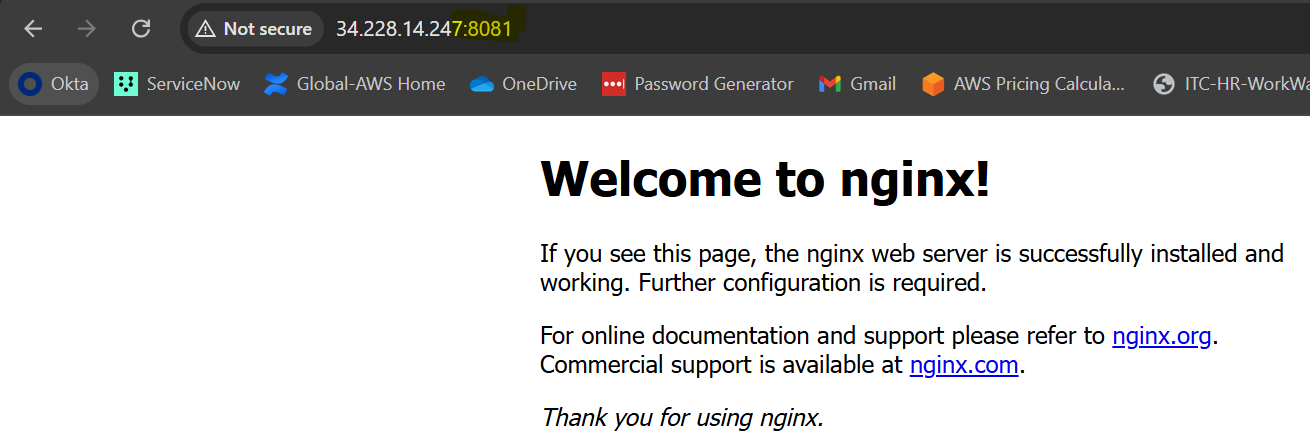




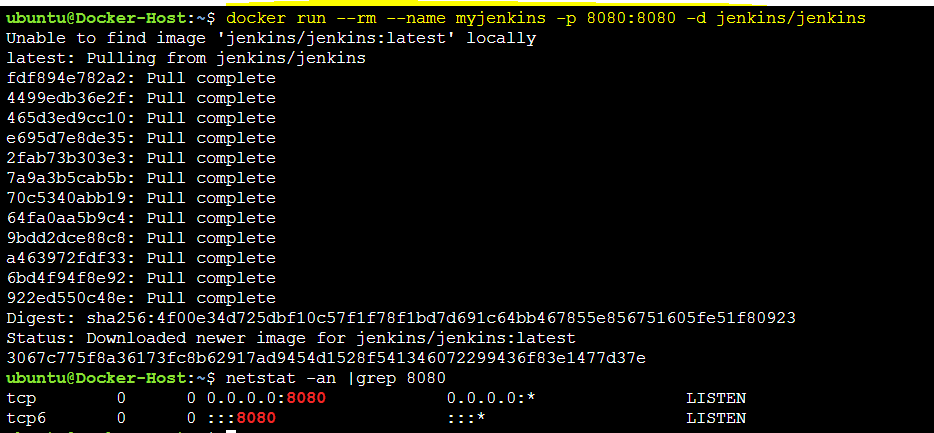


$ docker run --rm --name mynginx -p 8081:80 -d nginx



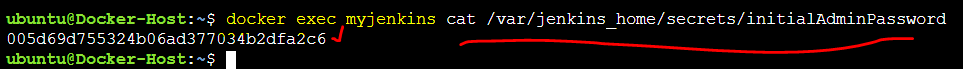


$ docker run --rm --name myjenkins -p 8080:8080 -d jenkins/jenkins



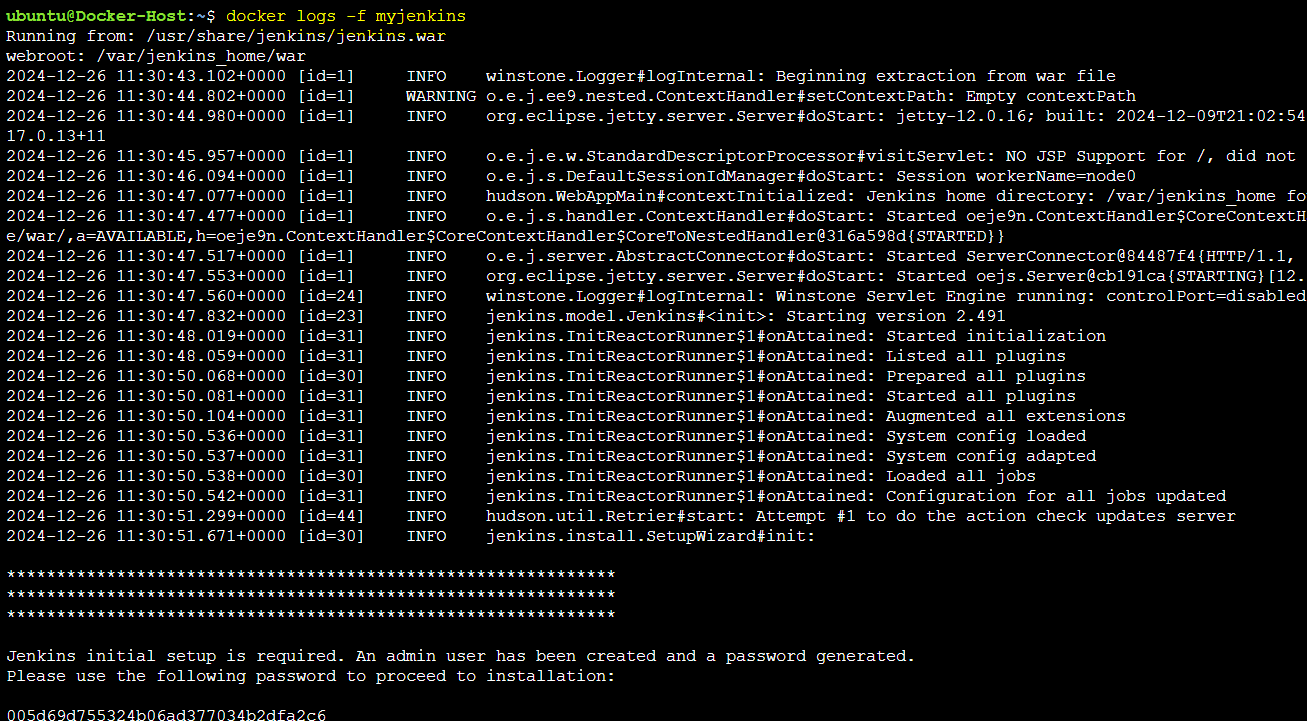


You can go into the container to get the password.



You check what is happening in the container without login [Detached mode]

**$ docker logs -f <cont-name>**



**Note:** If you above example only one host with different applications and ports. Just need to assign a port while creating the container.

**Docker Volumes**

If you want to access some data outside the container or shared we have an option called volumes.

To access that volume in the container we need to map that volume with the host machine in any folder. It will become the two-way sharing.

Like host to Container and container to host.

**IMP** - Volumes are used to share the data between one container to the host or one container to another container via the host.

The folder mapping can only happen while creating the container not after created the container.

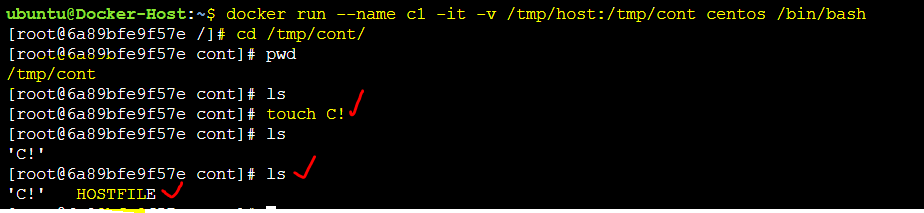
$ docker run --name <Cname> -it|-d -p <HP>:<CP> -v <**HD**>:<**CD**> <Image> <StartupCMD>

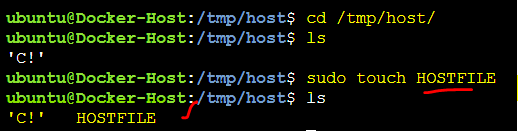
**HD** - Host Directory

**CD** - Container Directory

If the folder is not available then docker will create it while creating the container.

Create a file in both container and Host to check the file sharing /Volume mapping.





**IMP –**

If you stop the container also volume will be available.

If you remove the container also the volume will be available because a copy will exist in the host.

