Introduction to Docker

Docker is an open-source platform that facilitates the development, deployment, and management of applications in lightweight, portable containers. These containers are self-sufficient, encapsulating everything needed to run an application, including the code, runtime, libraries, and system tools. Docker provides a standardized way to package and distribute applications, making it easier to ensure consistency across different environments, from development to production.

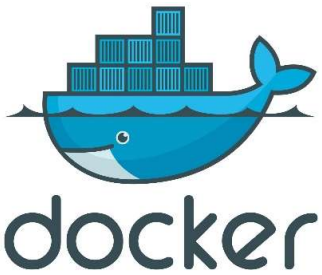
In essence, Docker allows you to create, deploy, and run applications in isolated containers, which can run consistently on any system that supports Docker, regardless of its underlying infrastructure.

Why Use Docker?

1. **Portability:** Docker containers are designed to be consistent across various environments, whether it's a developer's laptop, a testing server, or a production data center. This portability eliminates the classic "it works on my machine" problem, ensuring that applications behave the same way everywhere.
2. **Isolation:** Containers provide process and resource isolation, allowing multiple applications to run on the same host without interfering with each other. This isolation enhances security and minimizes conflicts between dependencies.
3. **Efficiency:** Docker containers are lightweight and share the host system's kernel. This means they have lower overhead compared to traditional virtual machines (VMs). As a result, you can run more containers on the same hardware, optimizing resource utilization.
4. **Rapid Deployment:** Docker simplifies the process of deploying and scaling applications. Containers can be started or stopped quickly, reducing the time it takes to get new features or updates into production.

5. **Version Control:** Docker enables version control for your application and its dependencies. You can define your application's environment in a Dockerfile, making it easy to reproduce specific configurations and roll back to previous versions if needed.
6. **Ecosystem:** Docker has a vast ecosystem of tools and services that enhance its functionality. Docker Compose for defining multi-container applications, Docker Swarm and Kubernetes for orchestration, and Docker Hub for sharing and discovering container images are just a few examples.
7. **DevOps Integration:** Docker is a fundamental building block of the DevOps movement. It streamlines collaboration between development and operations teams by providing a common platform and consistent environments throughout the development and deployment lifecycle.

In this comprehensive Docker guide, we will delve deeper into these concepts, exploring how to create and manage containers, orchestrate containerized applications, and leverage Docker's powerful features for your development and deployment needs. Whether you are a developer, system administrator, or a DevOps engineer, understanding Docker is essential in today's rapidly evolving software landscape. Let's begin our journey into the world of Docker containers.



Docker Installation

Let's begin with the installation. I am using AWS cloud and Launched 1 Instance of Amazon Linux 2023. Login Inside the OS and For Installing the Docker we have command available:

- **yum install docker -y**

```
[root@ip-172-31-12-48 ~]# yum install docker -y
Last metadata expiration check: 3:43:21 ago on Wed Sep 20 07:03:08 2023.
Dependencies resolved.
================================================================================
 Package                          Architecture          Version
================================================================================
Installing:
 docker                           x86_64                20.10.25-1.amzn2023.0.1
```

After the installation, we need to start the Docker services. However, when we shut down the system, the Docker services will stop. To address this issue, we need to enable the Docker services permanently so that if the operating system is rebooted, the services will start automatically.

- **systemctl status docker**

```
[root@ip-172-31-12-48 ~]# systemctl status docker
○ docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
     Active: inactive (dead) since Wed 2023-09-20 10:45:58 UTC; 8min ago
   Duration: 1min 37.084s
TriggeredBy: ○ docker.socket
       Docs: https://docs.docker.com
   Main PID: 2036 (code=exited, status=0/SUCCESS)
        CPU: 449ms
```

- **systemctl enable docker --now**

```
[root@ip-172-31-12-48 ~]# systemctl enable docker --now
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-12-48 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Wed 2023-09-20 10:56:29 UTC; 1s ago
TriggeredBy: ● docker.socket
```

- **docker version**

```
[root@ip-172-31-12-48 ~]# docker version
Client:
 Version:           20.10.25
 API version:       1.41
 Go version:        go1.19.9
 Git commit:        b82b9f3
 Built:             Wed Jul 12 19:37:13 2023
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
```

We can see the Docker is installed in my system with version 20.10.25

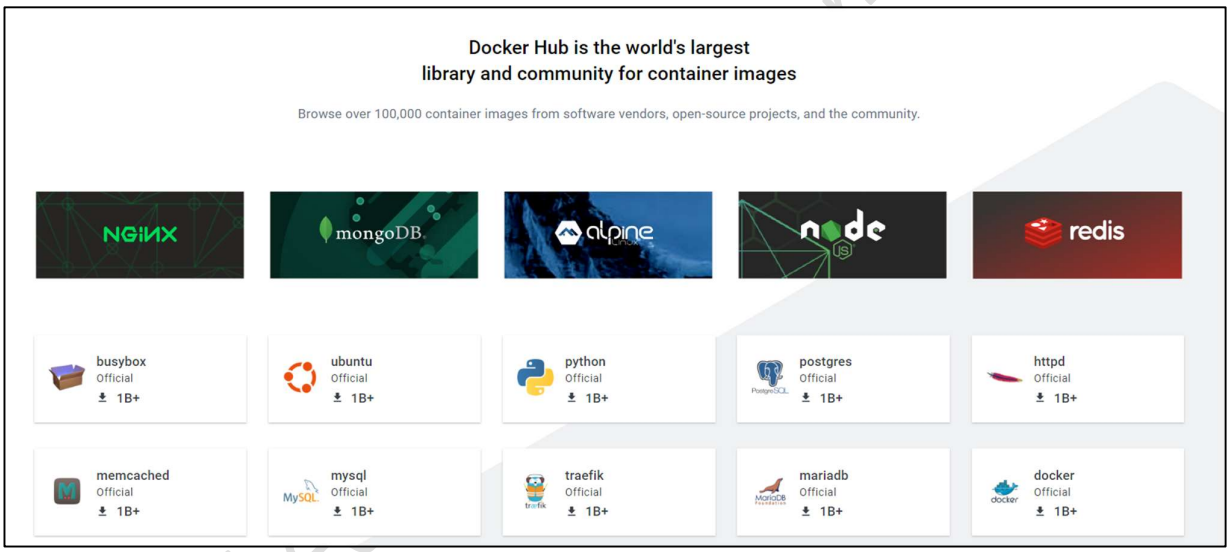
Managing Images

Docker images are like templates for creating containers. They contain all the necessary instructions and files for running an application. Think of them as a snapshot of an application's environment, allowing you to easily create and run consistent instances of that application in containers.

List Docker Images: > **docker images**

```
[root@ip-172-31-12-48 ~]#
[root@ip-172-31-12-48 ~]# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[root@ip-172-31-12-48 ~]#
```

Currently, no images are stored on my local operating system. We can obtain the image from its repository where all the images are stored. This repository is available to the public, and you can access it from: https://hub.docker.com/



I am familiar with Linux OS, so that's why I am going to download the CentOS Image. For downloading the image, we have command available:

- Syntax: **# docker pull Image-name:tag**
- Command: **docker pull centos**

Here I have not use any tags so be default it will pull the latest image of centos.

```
[root@ip-172-31-12-48 ~]# docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
[root@ip-172-31-12-48 ~]#
```

We can see that the image is Downloaded and we can verify it by:

- **docker images**

```
[root@ip-172-31-12-48 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
centos       latest    5d0da3dc9764   2 years ago   231MB
```

Last time, this image was not present here, but after downloading the image, it is now visible here.

If we don't requirement of this image then we can delete this image by using command:

- Syntax: **# docker rmi image-name**
- Command: **docker rmi centos:latest**

```
[root@ip-172-31-12-48 ~]# docker rmi centos:latest
Untagged: centos:latest
Untagged: centos@sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Deleted: sha256:5d0da3dc976460b72c77d94c8a1ad043720b0416bfc16c52c45d4847e53fadb6
Deleted: sha256:74ddd0ec08fa43d09f32636ba91a0a3053b02cb4627c35051aff89f853606b59
[root@ip-172-31-12-48 ~]# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[root@ip-172-31-12-48 ~]#
```

In the above command, I removed the CentOS image and also checked how many images are currently installed on my operating system. It shows nothing, which means there are no images available in my operating system.

Q. What is the Main purpose of docker image?

Ans. The main purpose of a Docker image is to package and contain all the necessary software and dependencies required to run a specific application or service. It encapsulates everything needed for the application to work consistently, making it easy to deploy and run on different computers or environments, without worrying about compatibility issues.

Q. If there is a requirement to launch a Container from CentOS. But there is no image is present in my Operating system. Then How we can launch the container?

Ans. The only way to launch the container is from the image. First, your operating system tries to find that image locally. If the image is not available on the local system, your operating system will attempt to connect with Docker Hub. If the image is found on Docker Hub, it will start downloading the image from Docker Hub and then launch the container from that image. If the image is not present on Docker Hub, you will receive an error message.

Let’s see this with practical example.

"When I run the command '**docker images**,' there are no images present on my OS. However, when I run the command '**docker run -it centos**,' this command tries to find the image locally. It is unable to find the image locally, it will begin downloading the image from Docker Hub, and then it starts the container."

```
[root@ip-172-31-12-48 ~]# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[root@ip-172-31-12-48 ~]#
[root@ip-172-31-12-48 ~]#
[root@ip-172-31-12-48 ~]# docker run -it centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
[root@93f474f0eec1 /]#
```

Managing Containers

A Docker container is like a small, isolated computer that runs a single application. It contains everything needed for the application to run, including code, libraries, and settings. Containers are efficient, lightweight, and portable, making it easy to run the same application consistently on different computers or servers.

The container is launched with the help of Docker image and from 1 Image we can launch as many containers as we want but how to check how many containers are currently, we have either in running state or stopped state? For this we have command available.

- **docker ps -a**

```
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
93f474f0eec1   centos    "/bin/bash"   14 minutes ago   Exited (0) 2 minutes ago              affectionate_sutherland
[root@ip-172-31-12-48 ~]#
```

There are several reasons why it's a good practice to remove containers from Docker:

- **Resource Management:** Containers consume system resources like CPU, memory, and disk space. Removing containers that are no longer needed helps free up these resources for other tasks and prevents resource exhaustion.
- **Isolation:** Containers are designed to be isolated from each other, but sometimes a container can be compromised or have issues. Removing a container ensures it can't affect other containers or the host system.
- **Security:** Leaving unused containers running can pose security risks. If a container has vulnerabilities or misconfigurations, it can be exploited by attackers. Removing unused containers helps mitigate these risks.
- **Clean Environment:** Removing containers helps maintain a clean and organized development or production environment. It's easier to manage and troubleshoot when only necessary containers are present.
- **Storage Management:** Docker stores container data, logs, and images on the host system. Over time, this can accumulate and consume significant disk space. Removing containers also removes their associated data, helping to manage storage.

- **docker rm Container-ID**
- **docker rm Container-Name**

```
[root@ip-172-31-12-48 ~]# docker rm 93f47
93f47
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-12-48 ~]#
```

Let's see about How we can launch a new Container from the Image. In my Host System I have downloaded 1 image of CentOS with latest version.

```
[root@ip-172-31-12-48 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
centos       latest    5d0da3dc9764   2 years ago    231MB
[root@ip-172-31-12-48 ~]#
```

And for Launching the container from this image we have command available.

- **docker create image-name:version**
- **docker create -arguments image-name:version**
- **docker create -arguments --name container-name image-name:version**

```
[root@ip-172-31-12-48 ~]# docker create centos:latest
0b3ea71a96c8f12cd2fe7943ea393431aad81846033dea45474897630bd4b898
[root@ip-172-31-12-48 ~]# docker create -it centos:latest
170af5b09898cee2575a3023b6a3dba15400a809e76bbf97b564cf3bde0c82e0
[root@ip-172-31-12-48 ~]# docker create -it --name test centos:latest
5252d1854259fb71433927bacffe0f72a025d120a6bbe92c63082101809eb498
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS    PORTS     NAMES
5252d1854259   centos:latest  "/bin/bash"   7 seconds ago    Created             test
170af5b09898   centos:latest  "/bin/bash"   20 seconds ago   Created             boring_driscoll
0b3ea71a96c8   centos:latest  "/bin/bash"   28 seconds ago   Created             condescending_benz
```

There are multiple ways to create the container. But giving a container a name when launching it from an image in Docker is a helpful practice for several reasons:

1. **Easy Identification:** Naming containers provides a human-readable identifier, making it much easier to distinguish and manage containers, especially when you have multiple containers running concurrently.
2. **Clarity in Logs and Status:** When you name containers, their names often appear in logs and status listings, making it clear which container corresponds to which application or service, simplifying troubleshooting and monitoring.
3. **Automation and Scripting:** When scripting or automating container management tasks, using named containers allows you to reference specific containers by name, making your scripts more understandable and maintainable.
4. **Network Configuration:** Naming containers can be particularly useful when dealing with networking. You can easily set up and manage network connections between containers using their names as references.
5. **Avoiding Conflicts:** Naming containers helps prevent naming conflicts. Docker container names must be unique within the Docker host, so giving containers descriptive names reduces the chance of accidental name collisions.
6. **Logging and Metrics:** Many logging and monitoring tools can collect data based on container names. Having meaningful names helps integrate Docker containers into your broader monitoring and observability systems.

We are Launching container for some purpose like My requirement is to run the date command in the container for that I need to go inside the container, and we have command available for this.

- **docker attach container-name**
- **docker attach container-Id**

```
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS    PORTS     NAMES
5252d1854259   centos:latest  "/bin/bash"   7 seconds ago    Created             test
170af5b09898   centos:latest  "/bin/bash"   20 seconds ago   Created             boring_driscoll
0b3ea71a96c8   centos:latest  "/bin/bash"   28 seconds ago   Created             condescending_benz
[root@ip-172-31-12-48 ~]# docker attach test
You cannot attach to a stopped container, start it first
```

Here we are receiving the error message that for Going inside the container we need to start it first and for starting the container we need to use this command:

- **docker start Container-name**
- **docker start Container-ID**

```
[root@ip-172-31-12-48 ~]# docker start 5252d
5252d
[root@ip-172-31-12-48 ~]# docker attach test
[root@5252d1854259 /]# date
Thu Sep 21 06:45:15 UTC 2023
[root@5252d1854259 /]#
```

Here I have use Container-Id to start the Container and then I have used Container name to go inside the container and after that I landed inside the container and then I run the **date** command and it shows the output.

If we are inside the container and we need to shut down the Container, then we can use **exit** command to shut down OR if we are not inside the container then we need to shut down the container then we can use this command:

- **docker stop container-name**
- **docker stop container-ID**

```
[root@ip-172-31-12-48 ~]# docker start 5252d
5252d
[root@ip-172-31-12-48 ~]# docker attach test
[root@5252d1854259 /]# exit
exit
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                      PORTS     NAMES
5252d1854259   centos:latest  "/bin/bash"   16 minutes ago   Exited (0) 2 seconds ago              test
170af5b09898   centos:latest  "/bin/bash"   16 minutes ago   Created                               boring_driscoll
0b3ea71a96c8   centos:latest  "/bin/bash"   16 minutes ago   Created                               condescending_benz
[root@ip-172-31-12-48 ~]# docker start 5252d
5252d
[root@ip-172-31-12-48 ~]# docker stop 5252d
5252d
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                      PORTS     NAMES
5252d1854259   centos:latest  "/bin/bash"   17 minutes ago   Exited (0) 3 seconds ago              test
```

Suppose If we are using the container for training the Model for Machine learning algorithms and we know that if we are using the Machine Learning for training complex model then it requires lots of time. But that time I want to use my HOST OS, then in this case I can use this KEY COMBINATION to detach from the container's interactive session without stopping the container:

- **CTRL + P + Q**

```
[root@ip-172-31-12-48 ~]# docker run -it --name machine_learning centos
[root@f813f4734a3f /]# sleep 90
[root@ip-172-31-12-48 ~]#
[root@ip-172-31-12-48 ~]#
```

Till now we have learnt about how to create the container using **docker create** command and how to go inside the container using **docker attach** command but there is a very simple way to go inside the container while launching the container.

- **docker run -it --name container-name image-name:tag**

```
[root@ip-172-31-12-48 ~]# docker run -it --name test2 centos
[root@9b2e3fcf65b2 /]# date
Thu Sep 21 09:13:19 UTC 2023
[root@9b2e3fcf65b2 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin
[root@9b2e3fcf65b2 /]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.3  12052  3160 pts/0    Ss   09:13   0:00 /bin/bash
root          17  0.0  0.3  47588  3236 pts/0    R+   09:13   0:00 ps -aux
[root@9b2e3fcf65b2 /]# exit
exit
[root@ip-172-31-12-48 ~]#
```

The **docker ps** command only shows the container which is in running state

```
[root@ip-172-31-12-48 ~]# docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS          PORTS     NAMES
5252d1854259   centos:latest  "/bin/bash"   3 hours ago   Up 22 seconds             test
170af5b09898   centos:latest  "/bin/bash"   3 hours ago   Up 22 seconds             boring_driscoll
```

The **docker ps -a** command list the container which is in running state or stopped state.

```
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS                      PORTS     NAMES
9b2e3fcf65b2   centos         "/bin/bash"   3 minutes ago   Exited (0) 2 minutes ago              test2
c3d4c41bad67   centos         "/bin/bash"   5 minutes ago   Exited (127) 3 minutes ago            test1
f813f4734a3f   centos         "/bin/bash"   2 hours ago     Exited (0) 2 hours ago                machine_learning
5252d1854259   centos:latest  "/bin/bash"   3 hours ago     Up 24 seconds                         test
170af5b09898   centos:latest  "/bin/bash"   3 hours ago     Up 24 seconds                         boring_driscoll
0b3ea71a96c8   centos:latest  "/bin/bash"   3 hours ago     Created                               condescending_benz
```

As a developer, you need to use multiple containers everyday but if you need to remove all containers after use then How can you perform this thing in one go. Here we are using some small Trick from Linux:

- **docker ps -aq**

This command will help you to retrieve all the container ID in one go

```
[root@ip-172-31-12-48 ~]# docker ps -aq
9b2e3fcf65b2
c3d4c41bad67
f813f4734a3f
5252d1854259
170af5b09898
0b3ea71a96c8
```

Now, we are going to pass this output inside to **docker rm** command.

- **docker rm -f $(docker ps -aq)**

This command captures the output from docker ps -aq (which lists all container IDs) then uses it as input to forcibly remove each container one by one.

```
[root@ip-172-31-12-48 ~]# docker rm -f $(docker ps -aq)
9b2e3fcf65b2
c3d4c41bad67
f813f4734a3f
5252d1854259
170af5b09898
0b3ea71a96c8
[root@ip-172-31-12-48 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-12-48 ~]#
```