**DOCKER**

**Docker** is a platform that is used to **containerize** our software, using which we can easily build our applications and **package** them, with the dependencies required, into containers, and these containers are easily shipped to run on other machines.



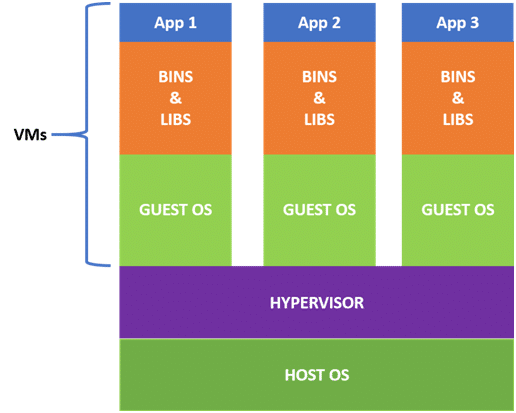
Docker simplifies the [DevOps Methodology](https://intellipaat.com/blog/what-is-devops/) by allowing developers to create templates called “images,” using which we can create lightweight virtual machines called “containers.”

**What is virtualization?**

Virtualization refers to importing a guest operating system on the host operating system and allowing developers to run multiple OS on different VMs while all of them run on the same host, thereby eliminating the need to provide extra hardware resources.

These virtual machines are being used in the industry in many ways:

* Enabling multiple operating systems on the same machine
* Cheaper than the previous methods due to less or compact infrastructure setup
* Easy to recover and do maintenance if there is any failure
* Faster provisioning of applications and resources required for tasks
* Increase in IT productivity, efficiency, and responsiveness



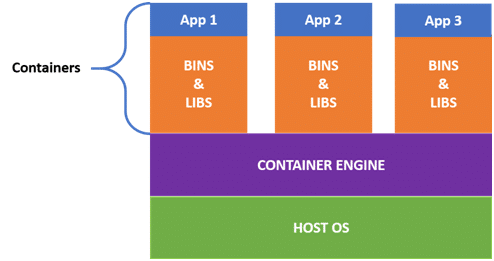
When we use VMs in virtualization, the bootup process takes a long time that affects efficiency in the case of real-time applications. In order to overcome such limitations, **containerization** was introduced.

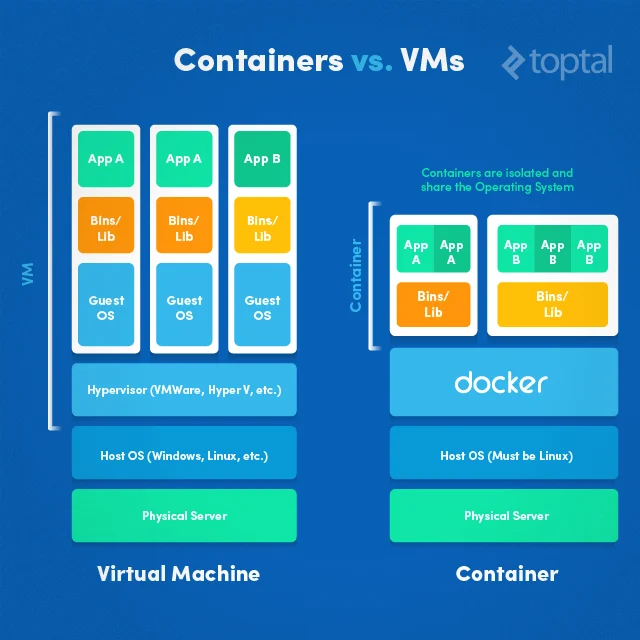
**What is containerization?**

**Containerization** (containers) is a lightweight virtualization technology acting as an alternative to hypervisor virtualization. Bundle any application in a container and run it without thinking of dependencies, libraries, and binaries .

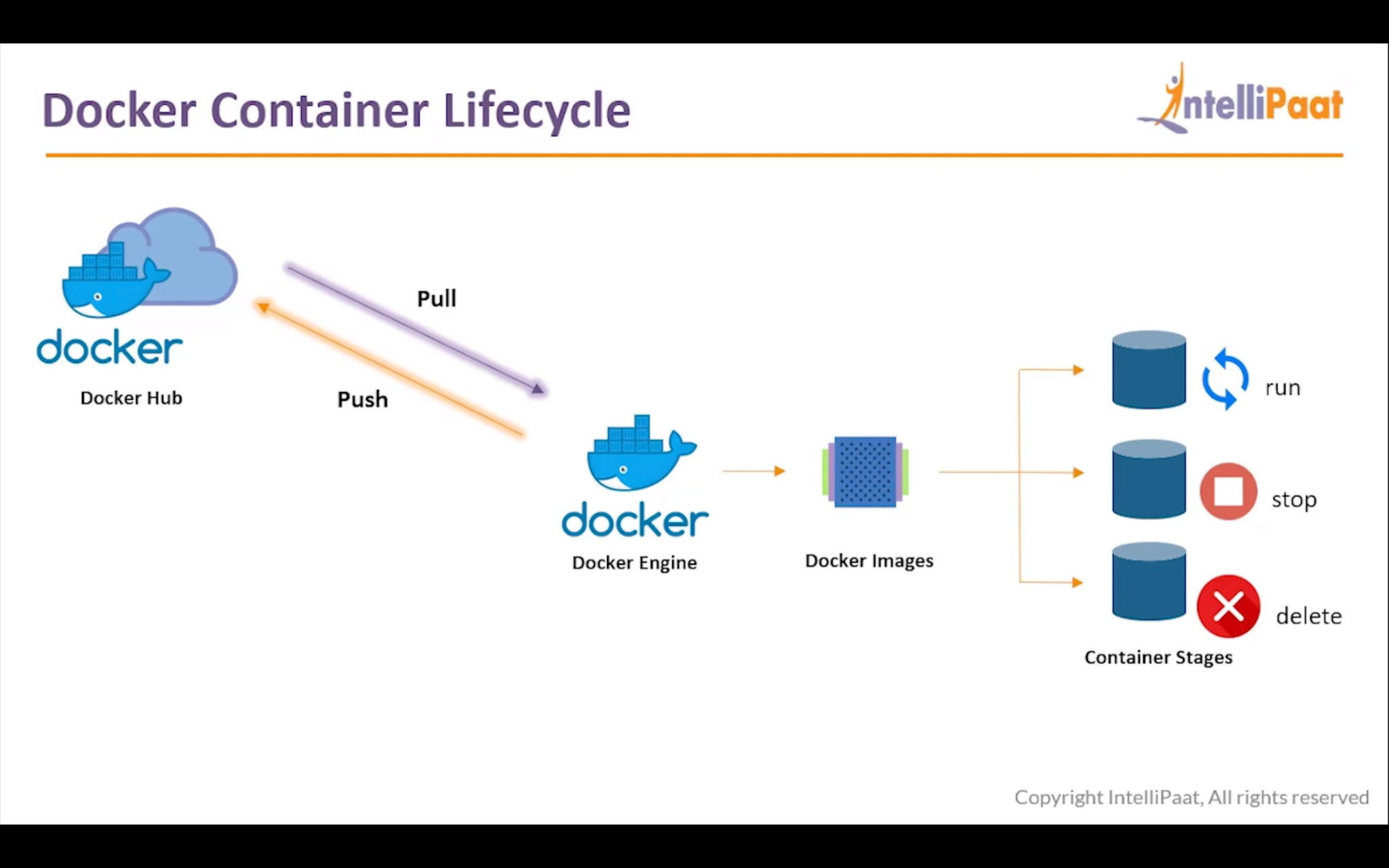
* Containers are small and lightweight as they share the same OS kernel.
* They do not take much time, only seconds, to boot up.
* They exhibit high performance with low resource utilization

| **Virtualization** | **Containerization** |
| --- | --- |
| Virtualizes hardware resources | Virtualizes only OS resources |
| Requires the complete OS installation for every VM | Installs the container only on a host OS |
| A kernel is installed for every virtualized OS | Uses only the kernel of the underlying host OS |
| Heavyweight | Lightweight |



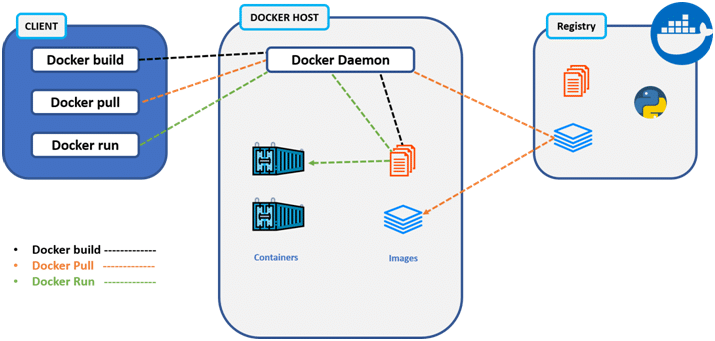


**Docker Container LifeCycle**

****

**Docker Architecture**

* Docker uses a **client-server** architecture.
* The **Docker client** consists of **Docker build,** **Docker pull,** and **Docker run**.
* The client approaches the **Docker daemon** that further helps in building, running, and distributing Docker containers.
* Docker client and Docker daemon can be operated on the same system. Otherwise, we can connect the Docker client to the remote Docker daemon.
* Both communicate with each other by using the REST API, over UNIX sockets or a network.



* Docker Client
* Docker Host
* Docker Registry

**Docker Client**

* It is the primary way for many Docker users to interact with Docker.
* It uses command-line utility or other tools that use Docker API to communicate with the Docker daemon.
* A Docker client can communicate with more than one Docker daemon.

**Docker Host**

In Docker host, we have a **Docker daemon** and **Docker objects** such as **containers** and **images**. First, let us understand the objects on the Docker host, then we will proceed toward the functioning of the Docker daemon.

* **Docker objects:**
  + **What is a Docker image?**
    - * A Docker image is a type of recipe or template that can be used for creating Docker containers. It includes steps for creating the necessary software.
  + **What is a Docker container?**
    - * A type of virtual machine that is created from the instructions found within the Docker image. It is a running instance of a Docker image that consists of the entire package required to run an application.
* **Docker daemon:**
  + Docker daemon helps in listening requests for the Docker API and in managing Docker objects such as images, containers, volumes, etc. Daemon issues to build an image based on a user’s input, and then saves it in the registry.
  + In case we do not want to create an image, then we can simply pull an image from the Docker hub, which might be built by some other user. In case we want to create a running instance of our Docker image, then we need to issue a run command that would create a Docker container.
  + A Docker daemon can communicate with other daemons to manage Docker services.

**Docker Registry**

* Docker registry is a repository for Docker images that are used for creating Docker containers.
* We can use a local or private registry or the Docker hub, which is the most popular social example of a Docker repository.

**COMMON DOCKER COMMANDS/ OPERATIONS**

Sudo apt-get update

Sudo apt-get install docker.io

1. **docker --version** (To check the version of docker is Installed)

Eg: **ubuntu@ip-172-31-31-118:~$ docker --version**

**Docker version 20.10.7, build 20.10.7-0ubuntu1~18.04.1**

1. **sudo docker pull <image-Name>** (helps to pull images from central Docker Repository)

Eg: **ubuntu@ip-172-31-31-118**:**~**$ **sudo docker pull ubuntu**

1. **sudo docker images** (Listing all docker images installed on your system)

Eg: **ubuntu@ip-172-31-31-118:~$ sudo docker images**

1. **sudo docker run <image-Name>** (Helps in running container with its image name)

Eg : **ubuntu@ip-172-31-31-118:~$ sudo docker run -it -d ubuntu**

**351079968e25cf676e14bb2fd300c738866513b0dc47226ecd3e5b5c6e344845**

**-it** → Make the terminal interactive

**-d** → Run the background container till we stop

1. **sudo docker ps** →Listing all the containers which are running on the system

Eg: **ubuntu@ip-172-31-31-118:~$ sudo docker ps**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

351079968e25 ubuntu "bash" 2 minutes ago Up 2 minutes thirsty\_mclean

1. **sudo docker stop <docker-ID> →** To stop the Container

Eg: **ubuntu@ip-172-31-31-118:~$ sudo docker stop 351079968e25**

351079968e25

1. **sudo docker ps -a** → To List all the container which are stopped
   1. Eg : **ubuntu@ip-172-31-31-118:~$ sudo docker ps -a**
2. **sudo docker exec -it <container-ID> bash** → For logging into /Accessing the Container

Eg: **ubuntu@ip-172-31-31-118:~$ sudo docker exec -it c11cdd7d6b44 bash**

**root@c11cdd7d6b44:/#**

**Bash → Current Terminal space which we are working**

==================================================================

Now we are inside the docker Container and Can perform any Operations :

1. **apt-get update**
2. **Exit** → To exit from the Container /Back to the Host OS

9. **sudo docker kill <container-ID> -**

If we can’t stop the container then we can use docker kill to kill the process. It is like we will force stop our PC by hard switching off our PC by clicking the off button.