Contents

[4. Containers 2](#_Toc114810357)

[What is docker 2](#_Toc114810358)

[What are virtual machines? 3](#_Toc114810359)

[Comparison Docker and virtual machines 3](#_Toc114810360)

[Docker Container Lifecycle Management 6](#_Toc114810361)

[Create Containers 7](#_Toc114810362)

[Start Container 7](#_Toc114810363)

[Run Container 7](#_Toc114810364)

[Pause Container 8](#_Toc114810365)

[Stop Container 8](#_Toc114810366)

[Delete Container 9](#_Toc114810367)

[Kill Container 10](#_Toc114810368)

[Difference between Docker Create, Docker Start And Docker Run 10](#_Toc114810369)

[Linux Containers - LXC and LXD 11](#_Toc114810370)

[LXC: 11](#_Toc114810371)

[LXD: 11](#_Toc114810372)

[Cloud Orchestration : 11](#_Toc114810373)

[Important Facts : 12](#_Toc114810374)

[Advantages : 12](#_Toc114810375)

[Conclusion : 12](#_Toc114810376)

# 4. Containers

## What is docker

(refer attach file)

Docker is a set of platforms as a service (PaaS) product that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc

What is Docker?

Organizations in today’s world look forward to transforming their business digitally but are constrained by the diverse portfolio of applications, cloud, and on-premises-based infrastructure. Docker solves this obstacle of every organization with a container platform that brings traditional applications and microservices built on Windows, Linux, and mainframe into an automated and secure supply chain.

Docker is a software development tool and a virtualization technology that makes it easy to develop, deploy, and manage applications by using containers. Container refers to a lightweight, stand-alone, executable package of a piece of software that contains all the libraries, configuration files, dependencies, and other necessary parts to operate the application.

In other words, applications run the same irrespective of where they are and what machine they are running on because the container provides the environment throughout the software development life cycle of the application. Since containers are isolated, they provide security, thus allowing multiple containers to run simultaneously on the given host. Also, containers are lightweight because they do not require an extra load of a hypervisor. A hypervisor is a guest operating system like VMWare or VirtualBox, but instead, containers run directly within the host’s machine kernel.

Containers provide the following benefits:

Reduced IT management resources

Reduced size of snapshots

Quicker spinning up apps

Reduced and simplified security updates

Less code to transfer, migrate, and upload workloads

To start your Docker journey, check out Cloud Academy’s [Docker in Depth](https://cloudacademy.com/learning-paths/cloud-academy-docker-in-depth-129/) Learning Path.

## What are virtual machines?

Virtual machines, on the other hand, are created to perform tasks that, if otherwise performed directly on the host environment, may prove to be risky. Virtual machines are isolated from the rest of the system; the software inside the virtual machine cannot tamper with the host computer. Therefore, implementing tasks such as accessing virus infected data and testing of operating systems are done using virtual machines. We can define a virtual machine as:

A virtual machine is a computer file or software usually termed as a guest, or an image that is created within a computing environment called the host.

A virtual machine is capable of performing tasks such as running applications and programs like a separate computer making them ideal for testing other operating systems like beta releases, creating operating system backups, and running software and applications. A host can have several virtual machines running at a specific time. Logfile, NVRAM setting file, [virtual disk file](https://searchvirtualdesktop.techtarget.com/definition/virtual-hard-disk-VHD), and configuration file are some of the key files that make up a virtual machine. Another sector where virtual machines are of great use is server virtualization. In server virtualization, a physical server is divided into multiple isolated and unique servers, thereby allowing each server to run its operating system independently. Each virtual machine provides its virtual hardware, such as CPUs, memory, network interfaces, hard drives, and other devices.

Virtual machines are broadly divided into two categories depending upon their use:

System Virtual Machines: A platform that allows multiple virtual machines, each running with its copy of the operating system to share the physical resources of the host system. Hypervisor, which is also a software layer, provides the virtualization technique. The hypervisor executes at the top of the operating system or the hardware alone.

Process Virtual Machine: Provides a platform-independent programming environment. The process virtual machine is designed to hide the information of the underlying hardware and operating system and allows the program to execute in the same manner on every given platform.

To learn more about virtual machines, check out Cloud Academy’s [Virtual Machines Overview](https://cloudacademy.com/course/70-532-exam-prep/virtual-machines-overview/) Course. If you don’t already have a Cloud Academy account, you can [sign up](https://cloudacademy.com/personal-trial/) for a free 7-day trial.

Although several virtual machines running at a time may sound efficient, it leads to unstable performance. As the guest OS would have its kernel, set of libraries and dependencies, this would take up a large chunk of system resources.

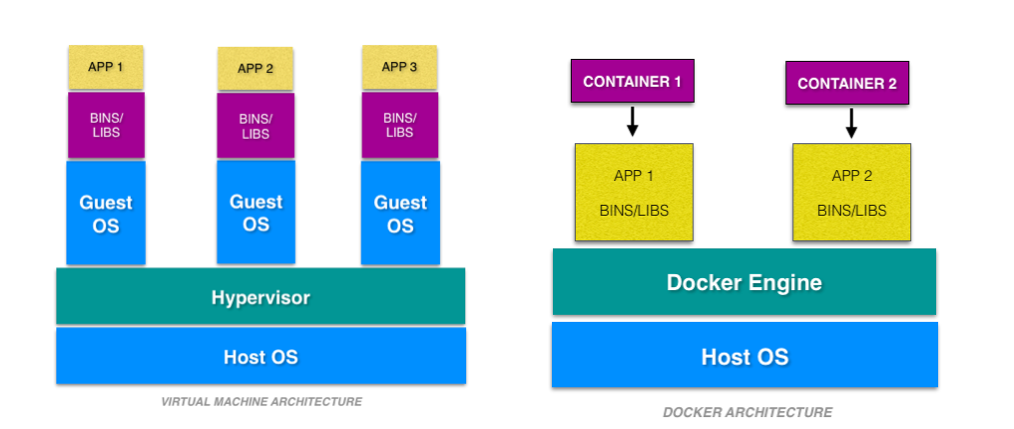
Other drawbacks include inefficient hypervisor and long boot uptime. The concept of Containerization overcomes these flaws. Docker is one such containerization platform.

## Comparison Docker and virtual machines

The following are the significant differences between Docker and virtual machines.

1. Docker vs. Virtual Machines: OS Support and Architecture

The main difference lies in their architecture, demonstrated below.



Virtual machines have host OS and the guest OS inside each VM. Guest OS can be any OS, like Linux or Windows, irrespective of host OS. In contrast, Docker containers host on a single physical server with a host OS, which shares among them. Sharing the host OS between containers makes them light and increases the boot time. Docker containers are considered suitable to run multiple applications over a single OS kernel; whereas, virtual machines are needed if the applications or services required to run on different OS.

2. Docker vs. Virtual Machines: Security

Virtual machines are stand-alone with their kernel and security features. Therefore, applications needing more privileges and security run on virtual machines.

On the flip side, providing root access to applications and running them with administrative premises is not recommended in the case of Docker containers because containers share the host kernel. The container technology has access to the kernel subsystems; as a result, a single infected application is capable of hacking the entire host system.

3. Docker vs. Virtual Machines: Portability

Virtual machines are isolated from their OS, and so, they are not ported across multiple platforms without incurring compatibility issues. At the development level, if an application is to be tested on different platforms, then Docker containers must be considered.

Docker containers packages are self-contained and can run applications in any environment, and since they don’t need a guest OS, they can be easily ported across different platforms. Docker containers can be easily deployed in servers since containers being lightweight can be started and stopped in very less time compared to virtual machines.

4. Docker vs. Virtual Machines: Performance

Virtual machines are more resource-intensive than Docker containers as the virtual machines need to load the entire OS to start. The lightweight architecture of Docker containers is less resource-intensive than virtual machines.

In the case of virtual machines, resources like CPU, memory, and I/O may not be allocated permanently to containers — unlike in the case of containers, where the resource usage with the load or traffic.

Scaling up and duplicating containers in simple and easy as compared to virtual machines because there is no need to install an operating system in them.

Apart from the major differences, some other differences are summarized below:

|  |  |  |
| --- | --- | --- |
|  | **Docker** | **Virtual Machines (VMs)** |
| **Boot-Time** | Boots in a few seconds. | It takes a few minutes for VMs to boot. |
| **Runs on** | Dockers make use of the execution engine. | VMs make use of the hypervisor. |
| **Memory Efficiency** | No space is needed to virtualize, hence less memory. | Requires entire OS to be loaded before starting the surface, so less efficient. |
| **Isolation** | Prone to adversities as no provisions for isolation systems. | Interference possibility is minimum because of the efficient isolation mechanism. |
| **Deployment** | Deploying is easy as only a single image, containerized can be used across all platforms. | Deployment is comparatively lengthy as separate instances are responsible for execution. |
| **Usage** | Docker has a complex usage mechanism consisting of both third party and docker managed tools. | Tools are easy to use and simpler to work with. |

**Which is a better choice?**

It won’t be fair to compare Docker and virtual machines since they are intended for different use. Docker, no doubt is gaining momentum these days, but they cannot be said to replace virtual machines. In spite of Docker gaining popularity, a virtual machine is a better choice in certain cases. Virtual machines are considered a suitable choice in a production environment, rather than Docker containers since they run on their own OS without being a threat to the host computer. But if the applications are to be tested then Docker is the choice to go for, as Docker provides different OS platforms for the thorough testing of the software or an application.

Furthermore, Docker containers use docker-engine instead of the hypervisor, like in virtual machines. As the host kernel in not shared, using docker-engine makes containers small, isolated, compatible, high performance-intensive and quickly responsive. Docker containers have comparatively low overhead as they have compatibility to share single kernel and application libraries. Organizations are making use of the hybrid approach mostly as the choice between virtual machines and Docker containers depends upon the kind of workload offered.

Also, not many digital operational companies rely on virtual machines as their primary choice and prefer migrating towards using containers as the deployment is comparatively lengthy and running microservices is also one of the major challenges it possesses. However, they are still some firms that prefer virtual machines over Dockers whereas companies who are interested in enterprise-grade security for their infrastructure prefer to make use of Dockers.

Finally, containers and Docker are not in conflict with virtual machines, they are both complementary tools for different workload and usage. Virtual machines are built for applications that are usually static and don’t change very often. Whereas, the Docker platform is built with a mindset to be more flexible so that containers can be updated easily and frequently.

So, is Docker just hype or just revolutionized — or is it replacing virtual machines? Comment your thoughts below or give further suggestions.

## **Docker Container Lifecycle Management**

There are different stages when we create a Docker container which is known as Docker Container Lifecycle. Some of the states are:

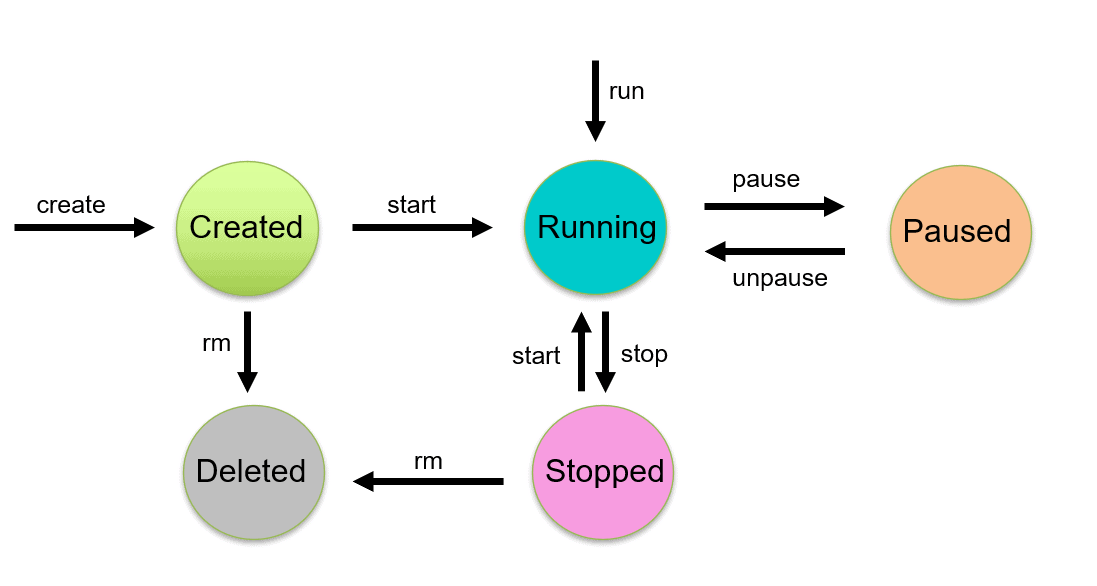
**Created**: A container that has been created but not started

**Running**: A container running with all its processes

**Paused**: A container whose processes have been paused

**Stopped**: A container whose processes have been stopped

**Deleted**: A container in a dead state



Commands in Docker Container Lifecycle Management

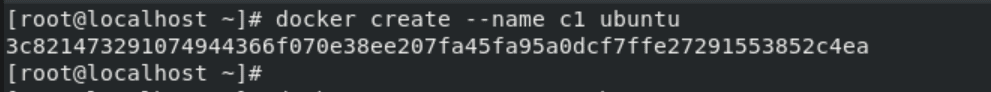
Managing the states of the Docker containers is called Docker Container Lifecycle Management. We have to assure that the containers are up and running, or destroyed if they are of no use anymore. For managing the Docker Lifecycle we have some common commands which are explained below.

**Check out:** [Docker & Certified Kubernetes Administrator (CKA)](https://k21academy.com/kubernetes/). A Kubernetes certification makes your resume look good and stand out from the competition. As companies rely more and more on Kubernetes, your expertise will be an immediate asset.

### Create Containers

Using the **docker create** command will create a new Docker container with the specified [**docker image**](https://k21academy.com/docker-kubernetes/docker-image-and-layer-overview-for-beginners/).

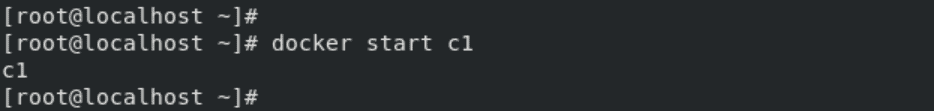
$ docker create --name <container name> <image name>



### Start Container

To start a stopped container, we can use the docker start command.

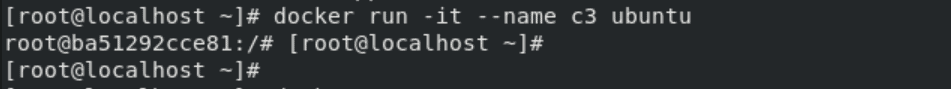
$ docker start <container name>



### **Run Container**

The **docker run** command will do the work of both “**docker create**” and “**docker start**” command. This command will create a new container and run the image in the newly created container.

$ docker run -it --name <container name> <image name>



**Check Out:**[Docker Tutorial for Beginners](https://k21academy.com/docker-kubernetes/docker-tutorial/).

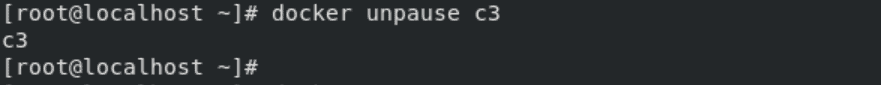
### Pause Container

If we want to pause the processes running inside the container, we can use the “**docker pause**” command.

$ docker pause <container name>

  
To unpause the container, use “**docker unpause**” command.

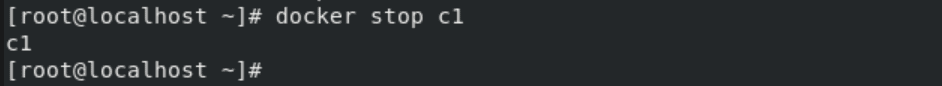
$ docker unpause <container name>



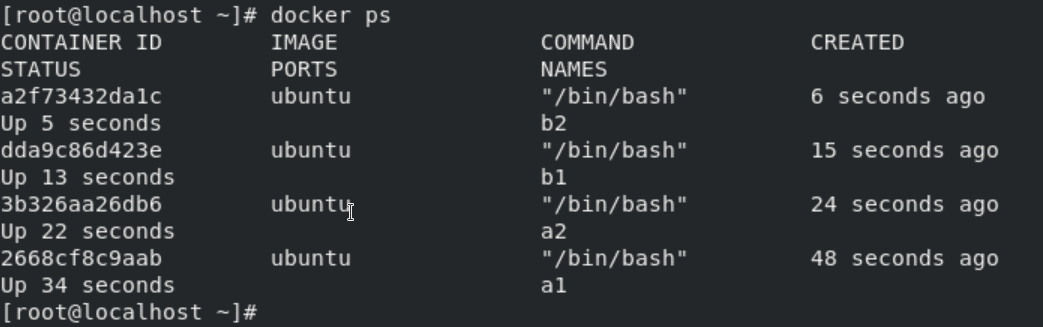
### Stop Container

Stopping a running Container means to stop all the processes running in that Container. Stopping does not mean killing or ending the process.

$ docker stop <container name>

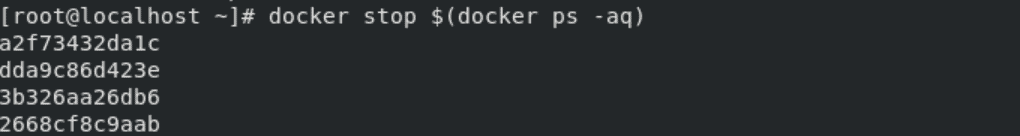
  
A stopped container can be made into the start state, which means all the processes inside the container will again start. When we do the docker stop command, the main process inside the container receives a **SIGTERM**signal. If you got confused about the term SIGTERM, then need not worry we will cover these signals later in the blog.

We can stop all the containers using a single command. In our case, 4 containers are running which you can see using the **docker ps** command.



To stop all the running containers we can use the following command:

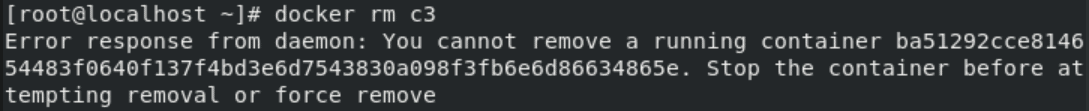
$ docker stop $(docker container ls –aq)



**Check Out:**[How to Fix Vulnerabilities in Docker Images](https://k21academy.com/docker-kubernetes/docker-image-vulnerabilities/).

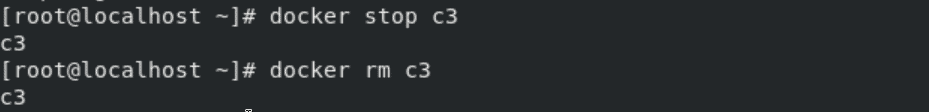
### Delete Container

Removing or deleting the container means destroying all the processes running inside the container and then deleting the Container. It’s preferred to destroy the container, only if present in the stopped state instead of forcefully destroying the running container.

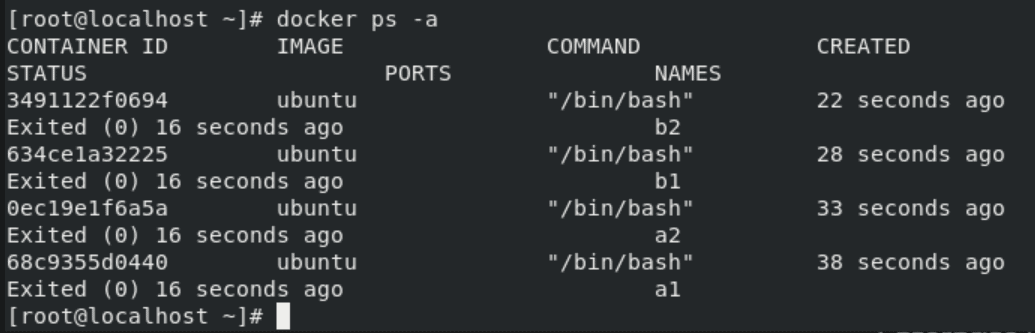
  
As we tried deleting a Container which was in running state, so the docker daemon throws an error. We have to first stop the container and delete it.

$ docker stop <container name>

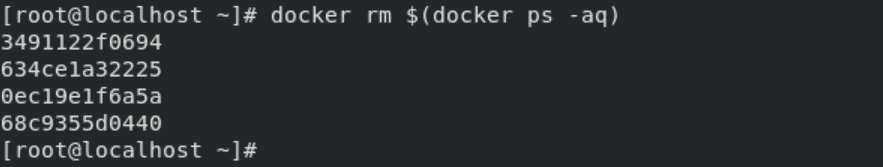
$ docker rm <container name>



We can delete or remove all containers with a single command only. In our example, 4 containers (not necessarily running) are there which you can see using the **docker ps -a** command.

We can see there are 4 containers which are not in the running state. Now we will delete all of them using a single command which is given below:

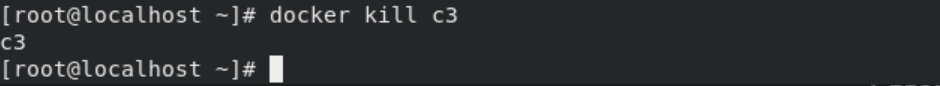
$ docker rm $(docker ps -aq)



### Kill Container

We can kill one or more running containers.

$ docker kill <container name>



**Also read:** [AKS Cluster](https://k21academy.com/docker-kubernetes/create-aks-cluster-step-by-step-procedure/) is a Kubernetes cluster, which is created on the Azure Kubernetes Service (AKS) by Microsoft is one of the leading managed K8s services.

## Difference between Docker Create, Docker Start And Docker Run

**Docker create command** creates a new container from the specified image. However, it will not run the container immediately.

**Docker start command** is used to start any stopped container. If we used the docker to create a command to create a container, then we can start it with this command.

**Docker run command** is a combination of creating and start as it creates a new container and starts it immediately. In fact, the docker run command can even pull an image from Docker Hub if it doesn’t find the mentioned image on your system.

**Also read**: [Container (Docker) vs Virtual Machines (VM)](https://k21academy.com/docker-kubernetes/docker-vs-virtual-machine/) to understand what is their difference.

Difference Between Docker Pause And Docker Stop container

The **docker pause** command suspends all processes in the specified containers. Traditionally, when suspending a process the **SIGSTOP** signal is used, which is observable by the process being suspended. Also, the memory portion would be there while the container is paused and again the memory is used when the container is resumed.

When we use the **docker stop** command, the main process inside the container receives **SIGTERM**signal, and after some time, **SIGKILL**. Also, it will release the memory used after the container is stopped.

**SIGTERM** is the signal of termination. The intention is to kill the process, gracefully or not, but to first allow it a chance to clean up.  
**SIGKILL** is the kill signal. The only behaviour is to kill the process, immediately.  
**SIGSTOP** is the pause signal. The only behaviour is to pause the process. The shell uses pausing (and its counterpart, resuming via **SIGCONT**) to implement job control.

**Check out** this article on [Docker Compose](https://k21academy.com/docker-kubernetes/docker-compose/)

Docker rm Vs. Docker Kill

**docker container rm**: Using docker rm, we can remove one or more containers from the host node and for doing container name or ID can be used.

**docker container kill**: The main process inside each container specified will be sent **SIGKILL** or any signal specified with option –signal.

## Linux Containers - LXC and LXD

### LXC:

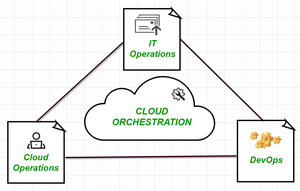
LXC—short for “Linux containers”, is a solution for virtualizing software at the operating system level within the Linux kernel. Unlike traditional hypervisors (think VMware, KVM and Hyper-V), LXC lets you run single applications in virtual environments, although you can also virtualize an entire operating system inside an LXC container

### LXD:

The simplest way to define LXD is to say it’s an extension of LXC. LXD also happens to be LXC’s main claim to fame, now that LXC has ceased to be important for Docker and CoreOS.

The more technical way to define LXD is to describe it as a REST API that connects to libxlc, the LXC software library. LXD, which is written in Go, creates a system daemon that apps can access locally using a Unix socket, or over the network via HTTPS.

Cloud Orchestration :  
Cloud Orchestration can be defined as the coordination, arrangement, or end-to-end automation of the deployment of services in a cloud-based environment. It introduces and enforces a workflow for automated activities of various processes to deliver the desired service to its client. Cloud Orchestration tools take full usage of [IaaS](https://www.geeksforgeeks.org/difference-between-iaas-paas-and-saas/) services providers to reach out to fully automated deployment, deleting the manual processes which are considered as time-consuming. Some Orchestration tools are Terraform, [Ansible](https://www.geeksforgeeks.org/ansible-introduction-and-installation-in-linux/) [AWS](https://www.geeksforgeeks.org/aws-types-of-databases/) Cloud Formation, etc.



Cloud-Based Infrastructure

### Important Facts :

It uses the programmed technique to manage overall the interconnections and interactions among the deployed workflows whether it is public or private cloud-based infrastructure.

Orchestrated solutions through cloud managers can be considered with a configuration of self-service resources to provide a faster service cycle than the older approach of manual configuration services based on individual requests.

This platform can integrate permission checks and security. It standardizes templates and enforces security practices, enabling the admin to review and improve already present automation scripts.

Preferably used to deploy or start servers, manage over the customer services network, create VMware’s and gain the access key of specific software on Cloud Model.

### Advantages :

The orchestration offers a systematic approach that leads to the increment of the automation benefits of agility and reduces costs.

It allows businesses to accelerate the deployment of advanced applications and services with coherence and proper manageability. It also leverages a unified portal and cloud-inspired IT service model with full-stack automation and monitoring.

This offers flexibility and increases the overall speed of creation, deployment, and operation of various micro-services provided by disparate systems.

Conclusion :  
Cloud Orchestration is now becoming an interest of many budding and even well-to-do organizations, as each enterprise are shifting towards better, faster, and secure delivery of micro-services to their customers with the right review and self-service configurations. This will additionally release the personal involvement of the enterprises. Overall Orchestration eliminates the potential of errors introduced into provisioning, scaling, and various other cloud purposes. One can consider orchestration technique as the full-fledged future of software cloud-based technology.

# 8. Distributed File System (DFS) and Hadoop

## What is DFS (Distributed File System)?

A **Distributed File System (DFS)**as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.



The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System. A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created and this process is transparent for the clients.

 DFS has two components:

* **Location Transparency –**  
  Location Transparency achieves through the namespace component.
* **Redundancy –**  
  Redundancy is done through a file replication component.

In the case of failure and heavy load, these components together improve data availability by allowing the sharing of data in different locations to be logically grouped under one folder, which is known as the “DFS root”.

It is not necessary to use both the two components of DFS together, it is possible to use the namespace component without using the file replication component and it is perfectly possible to use the file replication component without using the namespace component between servers.

**File system replication:**

Early iterations of DFS made use of Microsoft’s File Replication Service (FRS), which allowed for straightforward file replication between servers. The most recent iterations of the whole file are distributed to all servers by FRS, which recognises new or updated files.

“DFS Replication” was developed by Windows Server 2003 R2 (DFSR). By only copying the portions of files that have changed and minimising network traffic with data compression, it helps to improve FRS. Additionally, it provides users with flexible configuration options to manage network traffic on a configurable schedule.

#### Features of DFS :

* **Transparency :**
  + **Structure transparency –**  
    There is no need for the client to know about the number or locations of file servers and the storage devices. Multiple file servers should be provided for performance, adaptability, and dependability.
  + **Access transparency –**  
    Both local and remote files should be accessible in the same manner. The file system should be automatically located on the accessed file and send it to the client’s side.
  + **Naming transparency –**  
    There should not be any hint in the name of the file to the location of the file. Once a name is given to the file, it should not be changed during transferring from one node to another.
  + **Replication transparency –**  
    If a file is copied on multiple nodes, both the copies of the file and their locations should be hidden from one node to another.
* **User mobility :**   
  It will automatically bring the user’s home directory to the node where the user logs in.
* **Performance :**   
  Performance is based on the average amount of time needed to convince the client requests. This time covers the CPU time + time taken to access secondary storage + network access time. It is advisable that the performance of the Distributed File System be similar to that of a centralized file system.
* **Simplicity and ease of use :**   
  The user interface of a file system should be simple and the number of commands in the file should be small.
* **High availability :**   
  A Distributed File System should be able to continue in case of any partial failures like a link failure, a node failure, or a storage drive crash.   
  A high authentic and adaptable distributed file system should have different and independent file servers for controlling different and independent storage devices.
* **Scalability :**   
  Since growing the network by adding new machines or joining two networks together is routine, the distributed system will inevitably grow over time. As a result, a good distributed file system should be built to scale quickly as the number of nodes and users in the system grows. Service should not be substantially disrupted as the number of nodes and users grows.
* **High reliability :**  
  The likelihood of data loss should be minimized as much as feasible in a suitable distributed file system. That is, because of the system’s unreliability, users should not feel forced to make backup copies of their files. Rather, a file system should create backup copies of key files that can be used if the originals are lost. Many file systems employ stable storage as a high-reliability strategy.
* **Data integrity :**  
  Multiple users frequently share a file system. The integrity of data saved in a shared file must be guaranteed by the file system. That is, concurrent access requests from many users who are competing for access to the same file must be correctly synchronized using a concurrency control method. Atomic transactions are a high-level concurrency management mechanism for data integrity that is frequently offered to users by a file system.
* **Security :**   
  A distributed file system should be secure so that its users may trust that their data will be kept private. To safeguard the information contained in the file system from unwanted & unauthorized access, security mechanisms must be implemented.
* **Heterogeneity :**  
  Heterogeneity in distributed systems is unavoidable as a result of huge scale. Users of heterogeneous distributed systems have the option of using multiple computer platforms for different purposes.

#### History :

The server component of the Distributed File System was initially introduced as an add-on feature. It was added to Windows NT 4.0 Server and was known as “DFS 4.1”. Then later on it was included as a standard component for all editions of Windows 2000 Server. Client-side support has been included in Windows NT 4.0 and also in later on version of Windows.

Linux kernels 2.6.14 and versions after it come with an SMB client VFS known as “cifs” which supports DFS. Mac OS X 10.7 (lion) and onwards supports Mac OS X DFS.

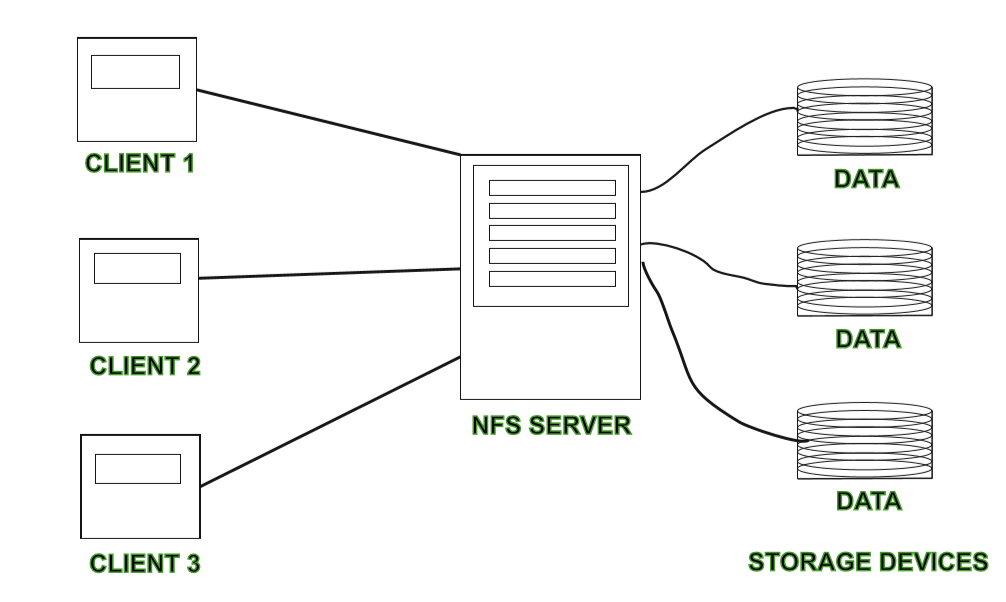
#### Applications :

* **NFS –**  
  NFS stands for Network File System. It is a client-server architecture that allows a computer user to view, store, and update files remotely. The protocol of NFS is one of the several distributed file system standards for Network-Attached Storage (NAS).
* **CIFS –**  
  CIFS stands for Common Internet File System. CIFS is an accent of SMB. That is, CIFS is an application of SIMB protocol, designed by Microsoft.
* **SMB –**  
  SMB stands for Server Message Block. It is a protocol for sharing a file and was invented by IMB. The SMB protocol was created to allow computers to perform read and write operations on files to a remote host over a Local Area Network (LAN). The directories present in the remote host can be accessed via SMB and are called as “shares”.
* **Hadoop –**  
  Hadoop is a group of open-source software services. It gives a software framework for distributed storage and operating of big data using the MapReduce programming model. The core of Hadoop contains a storage part, known as Hadoop Distributed File System (HDFS), and an operating part which is a MapReduce programming model.
* **NetWare –**  
  NetWare is an abandon computer network operating system developed by Novell, Inc. It primarily used combined multitasking to run different services on a personal computer, using the IPX network protocol.

#### Working of DFS :

There are two ways in which DFS can be implemented:

* **Standalone DFS namespace –**  
  It allows only for those DFS roots that exist on the local computer and are not using Active Directory. A Standalone DFS can only be acquired on those computers on which it is created. It does not provide any fault liberation and cannot be linked to any other DFS. Standalone DFS roots are rarely come across because of their limited advantage.
* **Domain-based DFS namespace –**  
  It stores the configuration of DFS in Active Directory, creating the DFS namespace root accessible at **\\<domainname>\<dfsroot>** or **\\<FQDN>\<dfsroot>**



#### Advantages :

* DFS allows multiple user to access or store the data.
* It allows the data to be share remotely.
* It improved the availability of file, access time, and network efficiency.
* Improved the capacity to change the size of the data and also improves the ability to exchange the data.
* Distributed File System provides transparency of data even if server or disk fails.

#### Disadvantages :

* In Distributed File System nodes and connections needs to be secured therefore we can say that security is at stake.
* There is a possibility of lose of messages and data in the network while movement from one node to another.
* Database connection in case of Distributed File System is complicated.
* Also handling of the database is not easy in Distributed File System as compared to a single user system.
* There are chances that overloading will take place if all nodes tries to send data at once