What Is Docker & Docker Container ? A Deep Dive Into Docker !

edureka.co/blog/what-is-docker-container

November 15, 2016

Last updated on Sep 11,2020*87.2K Views*

Vineet Chaturvedi

-
- Bookmark

3 / 11 Blog from Docker

Before we go ahead, let me summarize the learning till now:

- Virtual Machines are slow and take a lot of time to boot.
- Containers are fast and boots quickly as it uses host operating system and shares the relevant libraries.
- Containers do not waste or block host resources unlike virtual machines.
- Containers have isolated libraries and binaries specific to the application they are running.
- Containers are handled by Containerization engine.
- Docker is one of the containerization platforms which can be used to create and run containers.

Now, after this recap, let me take you ahead.

What is Docker & Docker Container ?

What is Docker & why is Docker needed? – Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.

What is Container ? – Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to full-fill the requirement from an operating system point of view. Also, it could be an application oriented container like CakePHP container or a Tomcat-Ubuntu container etc.

Let's understand it with an example:

A company needs to develop a Java Application. In order to do so the developer will setup an environment with tomcat server installed in it. Once the application is developed, it needs to be tested by the tester. Now the tester will again set up tomcat environment from the scratch to test the application. Once the application testing is done, it will be deployed on the production server. Again the production needs an environment with tomcat installed on it, so that it can host the Java application. If you see the same tomcat environment setup is done thrice. There are some issues that I have listed below with this approach:

1) There is a loss of time and effort.

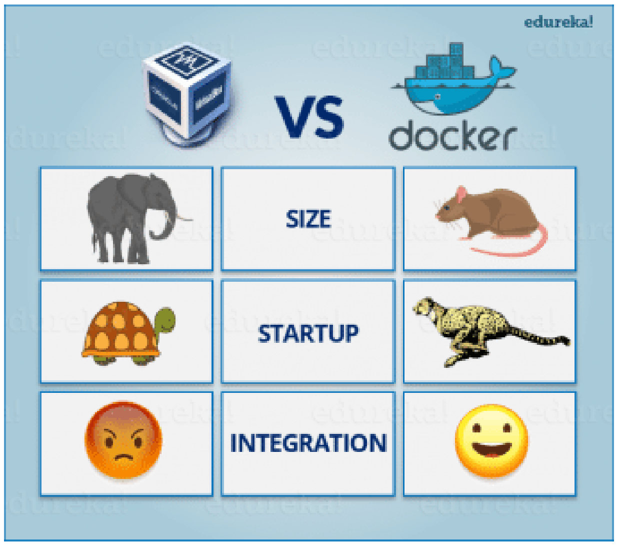
2) There could be a version mismatch in different setups i.e. the developer & tester may have installed tomcat 7, however the system admin installed tomcat 9 on the production server.

Now, I will show you how Docker container can be used to prevent this loss.

In this case, the developer will create a tomcat docker image (An Image is nothing but a blueprint to deploy multiple containers of the same configurations) using a base image like Ubuntu, which is already existing in Docker Hub (the Hub has some base images available for free) . Now this image can be used by the developer, the tester and the system admin to deploy the tomcat environment. This is how this container solves the problem.

I hope you are with me so far into the article . In case you have any further doubts, please feel to leave a comment, I will be glad to help you.

However, now you would think that this can be done using Virtual Machines as well. However, there is catch if you choose to use virtual machine. Let’s see a comparison between the two to understand this better.

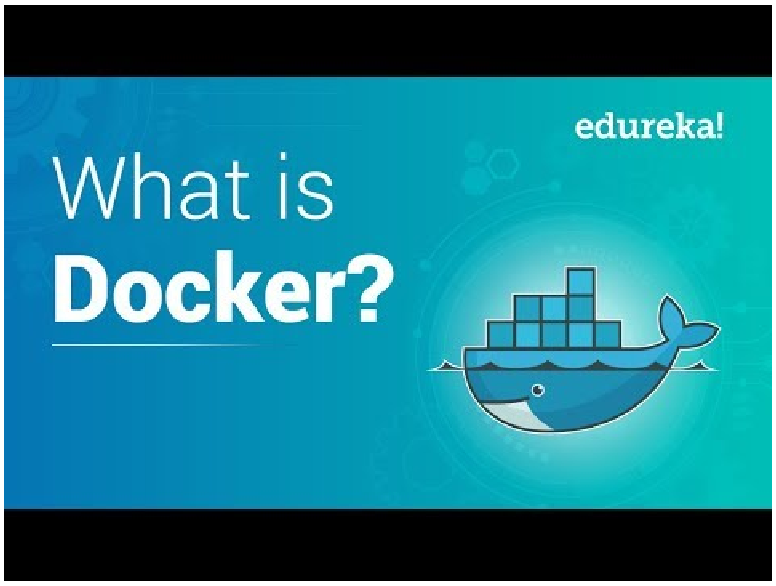


Let me take you through the above diagram. Virtual Machine and Docker Container are compared on the following three parameters:

- Size – This parameter will compare Virtual Machine & Docker Container on their resource they utilize.
- Startup – This parameter will compare on the basis of their boot time.
- Integration – This parameter will compare on their ability to integrate with other tools with ease.

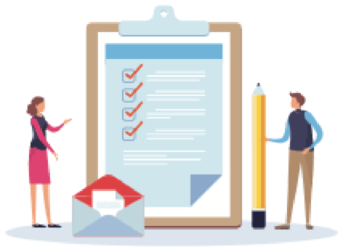
I will follow the above order in which parameters are listed. So first parameter would be “Size”.

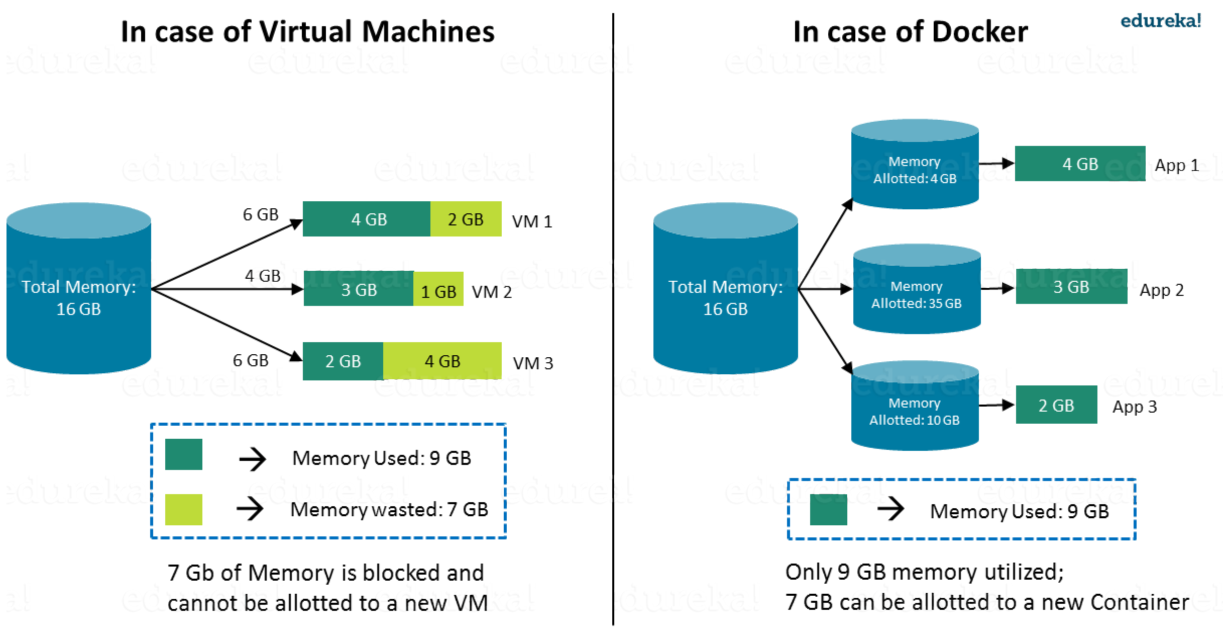
Check out the following video to know more about Docker Containers & why it is used.



Size

The following image explains how Virtual Machine and Docker Container utilize the resources allocated to them.





Consider a situation depicted in the above image. I have a host system with 16 Gigabytes of RAM and I have to run 3 Virtual Machines on it. To run the Virtual Machines in parallel, I need to divide my RAM among the Virtual Machines. Suppose I allocate it in the following way:

- 6 GB of RAM to my first VM,
- 4 GB of RAM to my second VM, and
- 6 GB to my third VM.

In this case, I will not be left with anymore RAM even though the usage is:

- My first VM uses only **4 GB** of RAM – Allotted **6 GB** – **2 GB** Unused & Blocked
- My second VM uses only **3 GB** of RAM – Allotted **4 GB** – **1 GB** Unused & Blocked
- My third VM uses only **2 GB** of RAM – Allotted **6 GB** – **4 GB** Unused & Blocked

This is because once a chunk of memory is allocated to a Virtual Machine, then that memory is blocked and cannot be re-allocated. I will be wasting **7 GB** (**2 GB** + **1 GB** + **4 GB**) of RAM in total and thus cannot setup a new Virtual Machine. This is a major issue because RAM is a costly hardware.

So, how can I avoid this problem?

If I use Docker, my CPU will allocates exactly the amount of memory that is required by the Container.

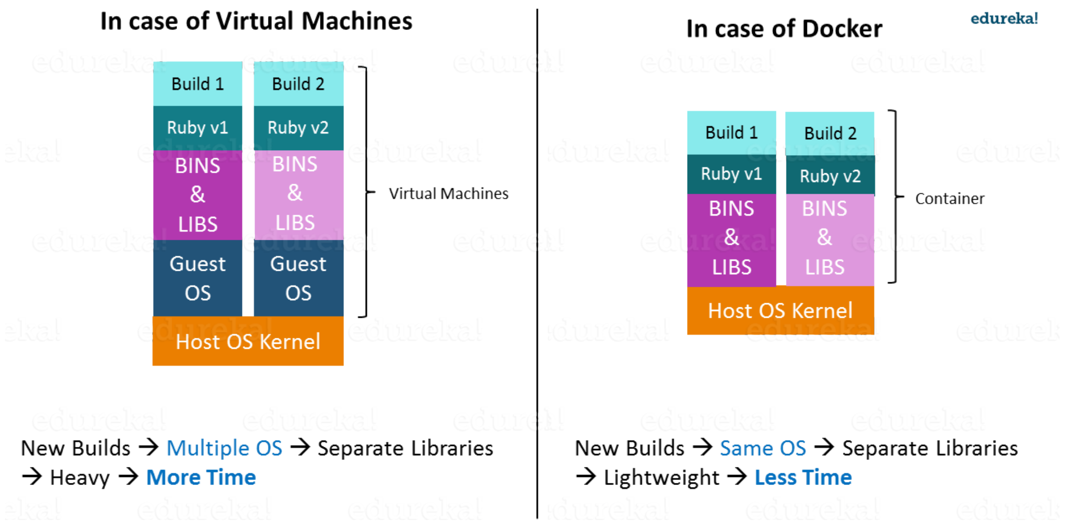
- My first container will use only **4 GB** of RAM – Allotted **4 GB** – **0 GB** Unused & Blocked

- My second container will use only **3 GB** of of RAM – Allotted **3 GB – 0 GB** Unused & Blocked
- My third container will use only **2 GB** of RAM – Allotted **2 GB – 0 GB** Unused & Blocked

Since there is no allocated memory (RAM) which is unused, I save **7 GB** (**16 – 4 – 3 – 2**) of RAM by using Docker Container. I can even create additional containers from the leftover RAM and increase my productivity.

So here Docker Container clearly wins over Virtual machine as I can efficiently use my resources as per my need.

Start-Up



When it comes to start-up, Virtual Machine takes a lot of time to boot up because the guest operating system needs to start from scratch, which will then load all the binaries and libraries. This is time consuming and will prove very costly at times when quick startup of applications is needed. In case of Docker Container, since the container runs on your host OS, you can save precious boot-up time. This is a clear advantage over Virtual Machine.

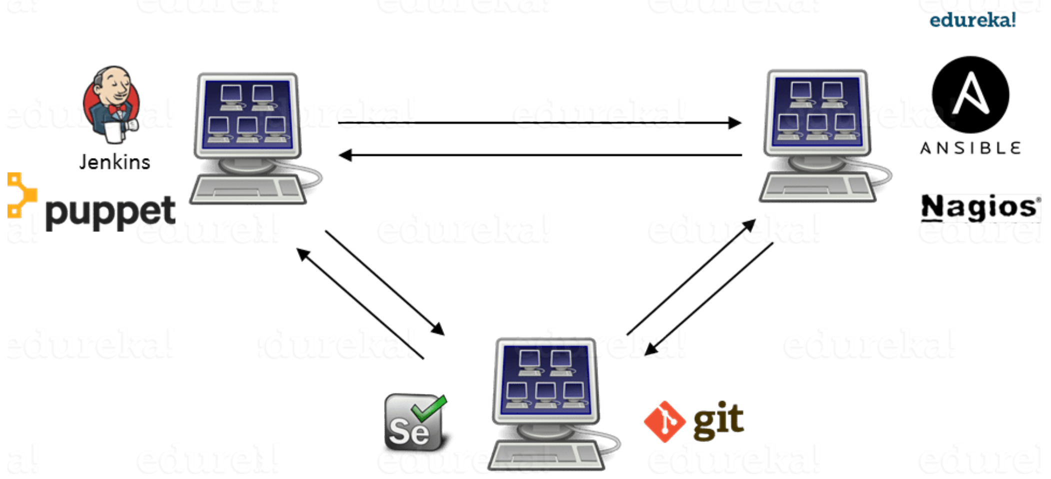
Consider a situation where I want to install two different versions of Ruby on my system. If I use Virtual Machine, I will need to set up 2 different Virtual Machines to run the different versions. Each of these will have its own set of binaries and libraries while running on different guest operating systems. Whereas if I use Docker Container, even though I will be creating 2 different containers where each container will have its own set of binaries and libraries, I will be running them on my host operating system. Running them straight on my Host operating system makes my Docker Containers lightweight and faster.

So Docker Container clearly wins again from Virtual Machine based on Startup parameter.

Now, finally let us consider the final parameter, i.e. Integration.

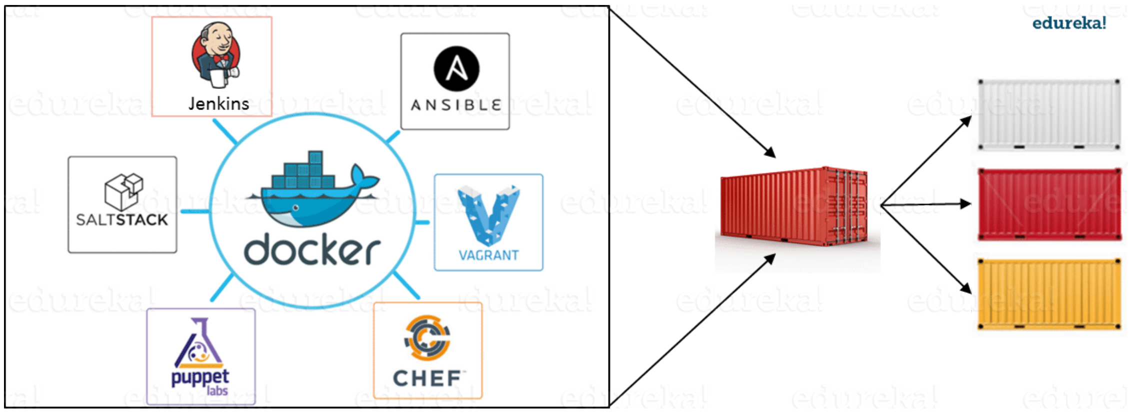
What about Integration?

Integration of different tools using Virtual Machine maybe possible, but even that possibility comes with a lot of complications.



I can have only a limited number of DevOps tools running in a Virtual Machine. As you can see in the image above, If I want many instances of Jenkins and Puppet, then I would need to spin up many Virtual Machines because each can have only one running instance of these tools. Setting up each VM brings with it, infrastructure problems. I will have the same problem if I decide to setup multiple instances of Ansible, Nagios, Selenium and Git. It will also be a hectic task to configure these tools in every VM.

This is where Docker comes to the rescue. Using Docker Container, we can set up many instances of Jenkins, Puppet, and many more, all running in the same container or running in different containers which can interact with one another by just running a few commands. I can also easily scale up by creating multiple copies of these containers. So configuring them will not be a problem.



DevOps Training

Next

To sum up, it won’t be an understatement to say that this is a more sensible option when compared to Virtual Machines.

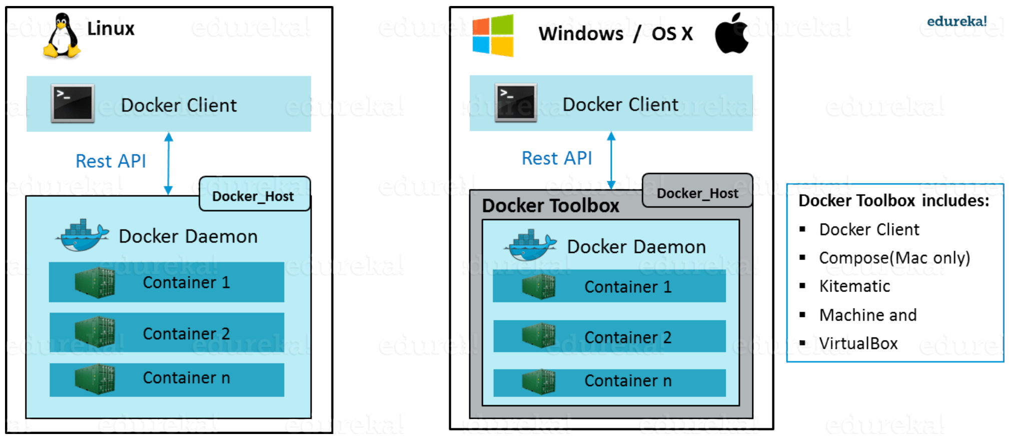
The former is designed to benefit both Developers and System Administrators, making it a part of many DevOps toolchains. Developers can write their code without worrying about the testing or the production environment and system administrators need not worry about infrastructure as Docker can easily scale up and scale down the number of systems for deploying on the servers.

What is Docker Engine?

Now I will take you through Docker Engine which is the heart of the system.

Docker Engine is simply the application that is installed on your host machine. It works like a client-server application which uses:

- A **server** which is a type of long-running program called a daemon process
- A command line interface (CLI) **client**
- REST API is used for communication between the CLI client and Docker Daemon



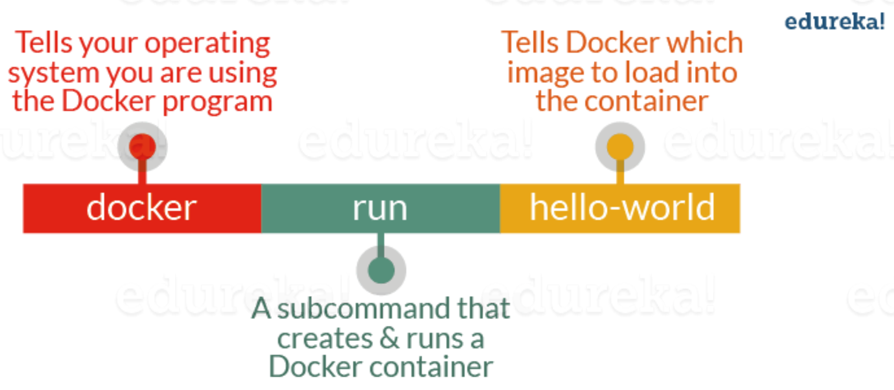
As per the above image, in a Linux Operating system, there is a client which can be accessed from the terminal and a Host which runs the Daemon. We build our images and run containers by passing commands from the CLI client to the Daemon.

However, in case of Windows/Mac there is an additional Toolbox component inside the Docker host. This Docker Toolbox is an installer to quickly and easily install and setup a Docker environment on your Windows/iOS. This Toolbox installs Docker Client, Machine, Compose (Mac only), Kitematic and VirtualBox.

Let’s now understand three important terms, i.e. **Docker Images**, **Docker Containers** and **Docker Registry**.

What is Docker Image?

Docker Image can be compared to a template which is used to create Docker Containers. They are the building blocks of a Docker Container. These Docker Images are created using the build command. These Read only templates are used for creating containers by using the run command. We will explore Docker commands in depth in the “Docker Commands blog”.

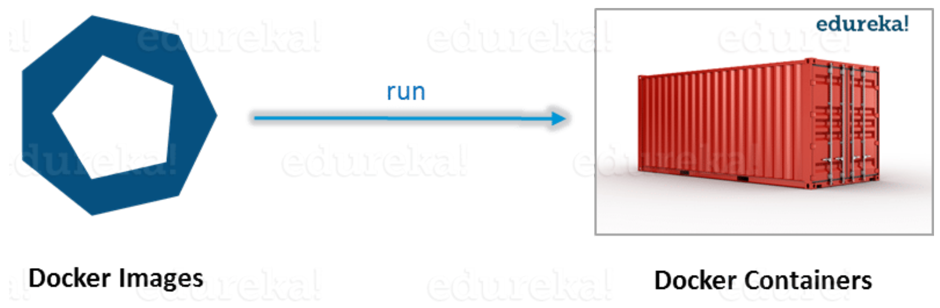

edureka!
Tells your operating system you are using the Docker program
Tells Docker which image to load into the container
docker
run
hello-world
A subcommand that creates & runs a Docker container

Docker lets people (or companies) create and share software through Docker images. Also, you don’t have to worry about whether your computer can run the software in a Docker image — a Docker container *can always run it*.

I can either use a ready-made docker image from docker-hub or create a new image as per my requirement. In the Docker Commands blog we will see how to create your own image.

What is Docker Container?

Docker Containers are the ready applications created from Docker Images. Or you can say they are running instances of the Images and they hold the entire package needed to run the application. This happens to be the ultimate utility of the technology.



What is Docker Registry?

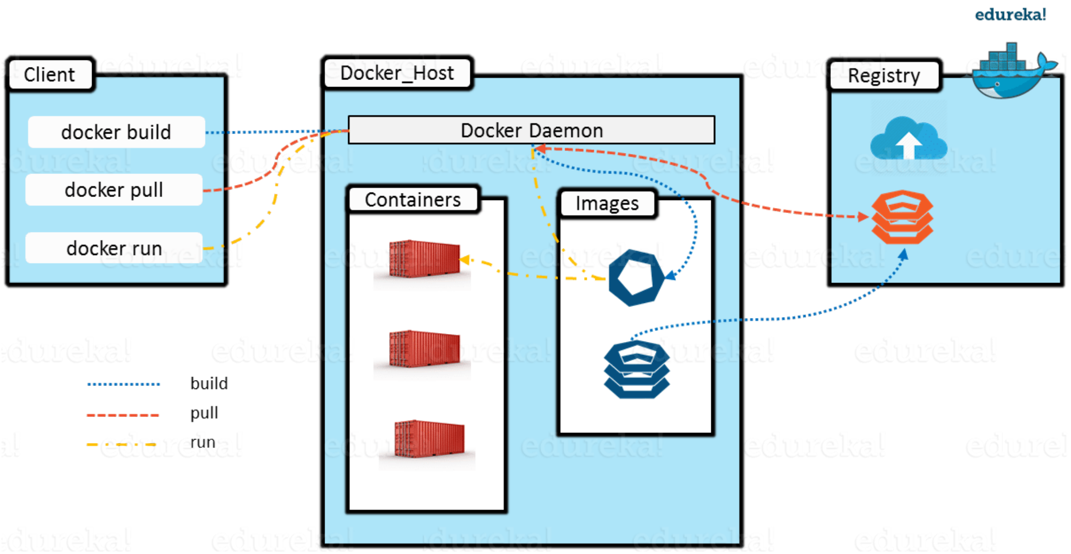
Finally, Docker Registry is where the Docker Images are stored. The Registry can be either a user’s local repository or a public repository like a Docker Hub allowing multiple users to collaborate in building an application. Even with multiple teams within the same organization can exchange or share containers by uploading them to the Docker Hub, which is a cloud repository similar to GitHub.

What is Docker Architecture?

Docker Architecture includes a Docker client – used to trigger Docker commands, a Docker Host – running the Docker Daemon and a Docker Registry – storing Docker Images. The Docker Daemon running within Docker Host is responsible for the images and containers.

Docker Certified Associate Training

Weekday / Weekend Batches See Batch Details



- To build a Docker Image, we can use the CLI (client) to issue a build command to the Docker Daemon (running on Docker_Host). The Daemon will then build an image based on our inputs and save it in the Registry, which can be either Docker hub or a local repository
- If we do not want to create an image, then we can just pull an image from the Docker hub, which would have been built by a different user
- Finally, if we have to create a running instance of my Docker image, we can issue a run command from the CLI, which will create a Container.

The aforementioned is a simple functionality of this technology!

I hope you enjoyed this blog. Now you are ready to get hands on experience with this. I will come up with the third blog on Docker Commands.

Now that you have understood what is DevOps, check out the DevOps training by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka DevOps Certification Training course helps learners gain expertise in various DevOps processes and tools such as Puppet, Jenkins, Nagios, Ansible, Chef, Saltstack and GIT for automating multiple steps in SDLC.

Got a question for us? Please mention it in the comments section and we will get back to you.

Thanks for your comment. You can certainly relate Docker containers to Ghost imaging however you will find docker container to be an evolved version and can be easily implemented using simple Docker Files. Using the Docker files you can define your images which can ultimately be used to launch docker containers. Also these containers are lightweight unlike Ghost imaging which will be heavy.

Do check out this video on Docker for more info: https://www.youtube.com/watch?v=h0NCZbHjIpY
You can also subscribe to the edureka channel as well for regular updates on DevOps videos.

Reply
Thank you for appreciating our work. We are glad that you liked it. Stay tuned for more such interesting updates. Cheers :)

Reply
can u explain more about namespaces and how they are handle by docker engine and kernal specially Cgroup and pid namespace

Reply
Is n’t Container concept seems to be coming from Ghost image.?
A good read that clarifies lots of basics.

Reply
This is very good information with detailed explanation.. Thank you.

Reply
Nice content with decent diagrams… Thank you….

Reply
Hey Sahi, thanks for checking out our blog! We’re glad you found it useful.
You might also like our tutorials here: https://www.youtube.com/edurekaIN.
Do subscribe to stay posted on upcoming blogs. Cheers!

Reply
Topics Covered

- DevOps (75 Blogs)
- Mastering Git and GitHub (9 Blogs)
- Docker (8 Blogs)
- DevOps Engineer Masters Program (4 Blogs)

SEE MORE

DevOps

DevOps Certification Training

- 81k Enrolled Learners
- Weekend/Weekday
- Live Class

Reviews

5 (32150)

AWS Certified DevOps Engineer Training

- 4k Enrolled Learners
- Weekend
- Live Class

Reviews

5 (1300)

Certified Kubernetes Administrator Exam Train ...

- 6k Enrolled Learners
- Weekend
- Live Class

Reviews

5 (2300)

Git and GitHub

- 4k Enrolled Learners
- Weekend
- Self Paced

Reviews

5 (1350)

Continuous Integration with Jenkins Certifica ...

- 8k Enrolled Learners
- Weekend
- Self Paced

Reviews

5 (3100)

Docker Certified Associate Training

- 6k Enrolled Learners
- Weekend
- Live Class

Reviews

5 (2100)

Comments

9 Comments

- Good work ✌️

  Reply

- “>

Reply

Join the discussion

CONTINUOUS
INTEGRATION
WITH JENKINS

DOCKER
CERTIFIED
ASSOCIATE TRAINING