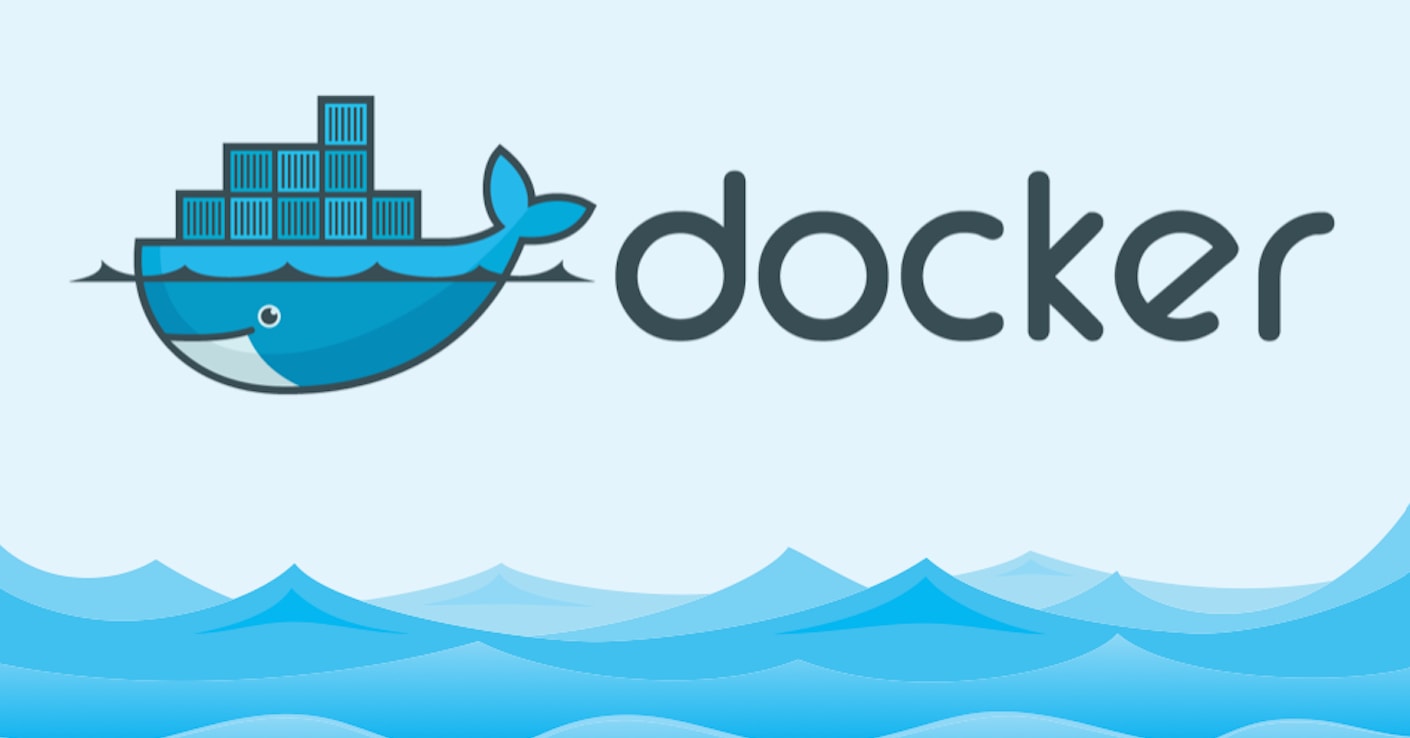
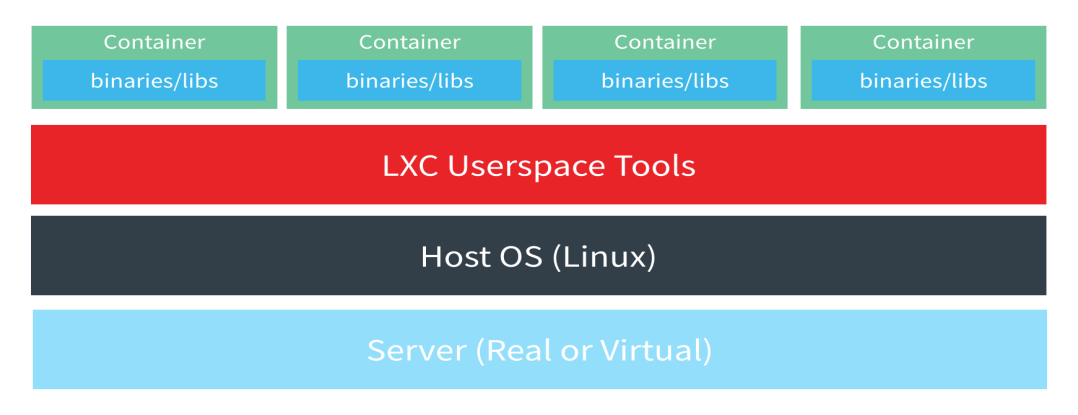
**What is Docker? How Does it Work?**



*Before understanding Docker, let’s have a look at Linux containers.*

## ****What is a Linux container?****

In a normal virtualized environment, one or more virtual machines run on top of a physical machine using a hypervisor like Xen, Hyper-V, etc. Containers, on the other hand, run in userspace on top of operating systems kernel. It can be called as OS-level virtualization. Each container will have its isolated user space and you can run multiple containers on a host, each having its own userspace. It means you can run different Linux systems (containers) on a single host. For example, you can run an RHEL and a SUSE container on an Ubuntu server. The Ubuntu Server can be a virtual machine or a physical host.



Note: you cannot run a windows container on a Linux host because there is no Linux Kernel support for Windows. You can read about Windows containers [from here](https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/)

Containers are isolated in a host using the two Linux kernel features called namespaces and control groups.

### ****Namespaces:****

There are six [namespaces](http://blog.dotcloud.com/under-the-hood-linux-kernels-on-dotcloud-part) in Linux

1. **pid namespace**: Responsible for isolating the process (PID: Process ID).
2. **net namespace**: It manages network interfaces (NET: Networking).
3. **ipc namespace**: It manages access to IPC resources (IPC: InterProcess Communication).
4. **mnt namespace**: Responsible for managing the filesystem mount points (MNT: Mount).
5. **uts namespace**: Isolates kernel and version identifiers. (UTS: Unix Timesharing System).

Using these namespaces a container can have its own network interfaces, IP address, etc. Each container will have its own namespace and the processes running inside that namespace will not have any privileges outside its namespace.

### ****Namespaces:****

There are six [namespaces](http://blog.dotcloud.com/under-the-hood-linux-kernels-on-dotcloud-part) in Linux

1. **pid namespace**: Responsible for isolating the process (PID: Process ID).
2. **net namespace**: It manages network interfaces (NET: Networking).
3. **ipc namespace**: It manages access to IPC resources (IPC: InterProcess Communication).
4. **mnt namespace**: Responsible for managing the filesystem mount points (MNT: Mount).
5. **uts namespace**: Isolates kernel and version identifiers. (UTS: Unix Timesharing System).

Using these namespaces a container can have its own network interfaces, IP address, etc. Each container will have its own namespace and the processes running inside that namespace will not have any privileges outside its namespace.

### ****Namespaces:****

There are six [namespaces](http://blog.dotcloud.com/under-the-hood-linux-kernels-on-dotcloud-part) in Linux

1. **pid namespace**: Responsible for isolating the process (PID: Process ID).
2. **net namespace**: It manages network interfaces (NET: Networking).
3. **ipc namespace**: It manages access to IPC resources (IPC: InterProcess Communication).
4. **mnt namespace**: Responsible for managing the filesystem mount points (MNT: Mount).
5. **uts namespace**: Isolates kernel and version identifiers. (UTS: Unix Timesharing System).

Using these namespaces a container can have its own network interfaces, IP address, etc. Each container will have its own namespace and the processes running inside that namespace will not have any privileges outside its namespace.

**Recommended Course:** [Docker Mastery: The Complete Toolset From a Docker Captain](https://devopscube.com/recommends/docker-devops-course/)

### ****Control groups:****

The resources used by a container is managed by Linux control groups. You can decide on how much CPU and memory resource a container should use using Linux control groups.

Container is not a new concept. Google has been using its own container technology in its Infrastructure for years. Solaris Zones, BSD Jails, LXC are the few Linux container technology that has been around for years. In this article, we will learn about Docker and see why Docker is very useful and different from other container technologies.

## ****What is Docker?****

Docker is a popular open-source project based on Linux containers. Docker is written in go and developed by Dotcloud (A PaaS Company). Docker is basically a container engine which uses the Linux Kernel features like namespaces and control groups to create containers on top of an operating system and automates application deployment on the container.

It provides a lightweight environment to run your application code. Docker has an efficient workflow for moving your application from developers laptop, test environment to production. It is incredibly fast and it can run on the host with compatible Linux Kernel. (Support Windows as well)

***Recommended eBook:***[*The Docker Book*](https://devopscube.com/go/docker-book/)

Docker uses Copy-on-write union file system for its backend storage. Whenever changes are made to a container, only the changes will be written to disk using copy on write model. Also, creating a container using Docker takes less than a second.

**Things you should know about Docker:**

1. Docker is not LXC
2. Docker is not a Virtual Machine Solution.
3. Docker is not a configuration management system and is not a replacement for chef, puppet, Ansible etc.
4. Docker is not a platform as a service technology.

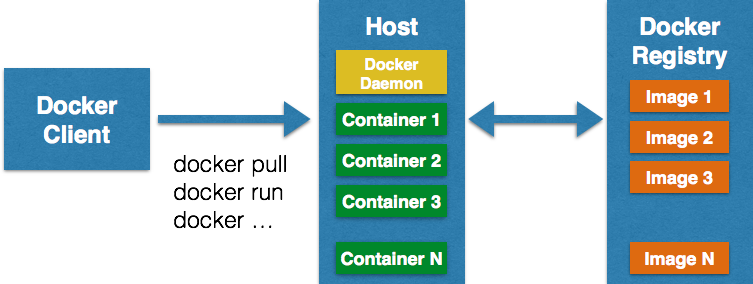
### ****Docker Components:****

Docker is composed of the following four components

1. Docker Client and Daemon.
2. Images
3. Docker registries
4. Containers

## How Does Docker Work?

Docker has a client-server architecture. Docker Daemon or server is responsible for all the actions that are related to containers. The daemon receives the commands from the Docker client through CLI or REST API’s. Docker client can be on the same host as a daemon or it can be present on any other host.



Images are the basic building blocks of Docker. Containers are built from images. Images can be configured with applications and used as a template for creating containers. It is organized in a layered fashion. Every change in an image is added as a layer on top of it.

**You Might Like**: [Kubernetes Tutorials For Beginners](https://devopscube.com/kubernetes-tutorials-beginners/)

**Docker registry** is a repository for Docker images. Using Docker registry, you can build and share images with your team. A registry can be public or private. Docker Inc provides a hosted registry service called Docker Hub. It allows you to upload and download images from a central location. If your repository is public, all your images can be accessed by other Docker hub users. You can also create a private registry in Docker Hub. Docker hub acts like git, where you can build your images locally on your laptop, commit it and then can be pushed to the Docker hub.

**Container** is the execution environment for Docker. Containers are created from images. It is a writable layer of the image. You can package your applications in a container, commit it and make it a golden image to build more containers from it. Two or more containers can be linked together to form tiered application architecture. Containers can be started, stopped, committed and terminated. If you terminate a container without committing it, all the changes made to the container will be lost.

### ****Conclusion:****

The best feature of Docker is collaboration. Docker images can be pushed to a repository and can be pulled down to any other host to run containers from that image. Moreover, Docker hub has thousands of images created by users and you can pull those images down to your hosts based on your application requirements. We will cover the more practical implementation of Docker in upcoming series of articles.

### You'll also like:

* [How To Link Docker Containers Across Hosts – The Ambassador Pattern](https://devopscube.com/how-to-link-docker-containers-across-hosts-the-ambassador-pattern/)
* [How to Setup Docker containers as Build Slaves for Jenkins](https://devopscube.com/docker-containers-as-build-slaves-jenkins/)
* [How to Provision Docker Hosts on Azure using Docker Machine](https://devopscube.com/provision-docker-hosts-on-azure/)
* [Getting Started With Docker : Working With Containers](https://devopscube.com/getting-started-with-docker-contianers/)
* [Docker Multi-Host Networking Tutorial – Using Consul](https://devopscube.com/docker-multi-host-networking-tutorial/)