**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 and light weight environment to run your application code. Docker has an efficient workflow for moving you application from developers laptop, test environment to production. Docker is incredibly fast and it can run on host with compatible Linux Kernel.

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 take 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.

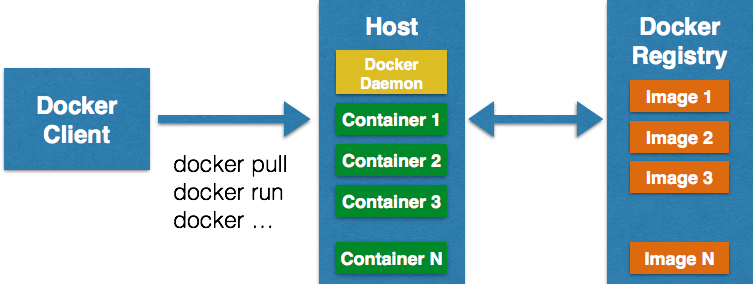
**Docker Components:**

Docker is composed of 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 though cli or REST API’s. Docker client can be on the same host as daemon or it can be present on any other host.

[](http://devopscube.com/wp-content/uploads/2014/12/docker-architecture-techtip39.png)

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. Images are organized in a layered manner. Every change in an image is added as layer on top of it.

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 in 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.

**BENFITS**

## ****Simplifying Configuration****

This is the primary use case Docker Inc. advocates. One of the big advantages of VMs is the ability to run any platform with its own config on top of your infrastructure. Docker provides this same capability without the overhead of a VM. It lets you put your environment and configuration into code and deploy it. The same Docker configuration can be used in a variety of environments. This decouples infrastructure requirements from the application environment.

## ****Code Pipeline Management****

The previous use case makes a large impact in managing the code pipeline. As the code travels from the developer’s machine to production, there are many different environments it has to go through to get there. Each of these may have minor differences. Docker provides a consistent environment to the application from dev through production, easing the code development and deployment pipeline.

## ****Debugging Capabilities****

Docker provides a lot of tools that are not necessarily specific to containers, but, they work well with the concept of containers. They also provide extremely useful functionality. This includes the ability to checkpoint and version containers, as well as to diff two containers. This can be immensely useful in fixing an application. You can find an example of this in our “[**Docker Saves the Day**](http://flux7.com/blogs/docker/docker-saves-the-day-at-flux7/)” post.

## ****Multi-Tenancy****

Yet another interesting use case of Docker is its use in multi-tenant applications, thereby avoiding major application rewrites. Our very own multi-tenant example is to develop quick and easy multi-tenancy for an IoT application. Code Bases for such multi-tenant applications are far more complicated, rigid and pretty much difficult to handle. Rearchitecting an application is not only time consuming, but also costs a lot of money.

Using Docker, it was easy and inexpensive to create isolated environments for running multiple instances of app tiers for each tenant. This was possible given the spin up speed of Docker environments and it’s effective diff command.

## ****Rapid Deployment****

containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly.

* **Simplified maintenance**

Docker reduces effort and risk of problems with application dependencies.

* ***Portability across machines***

an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transfered to another machine that runs **Docker**, and executed there without compatibility issues.

* ***Version control and component reuse***

you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.

* ***Sharing***

you can use a remote repository to share your container with others. Red Hat provides a registry for this purpose, and it is also possible to configure your own private repository.

* ***Lightweight footprint and minimal overhead***

Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers