What is Docker?

# Introduction

If you’re involved in the IT industry, chances are you’ve heard of [Docker](http://www.docker.com/). You may, however, not fully know what this particular technology is. Considering Docker is something you may very well want to use with your Linode, let’s take a moment to understand exactly what Docker is and why you should be using it.

## Containers

Docker is a tool that enables you to create, deploy, and manage containers. What are containers? In simplest form, a container is a lightweight, stand-alone package that contains everything needed to run an application (code, libraries, runtime, system settings, and dependencies). Each container will be deployed with its own CPU, memory, block I/O, and network resources, all without having to depend upon an individual kernel and operating system. And *that* is the biggest difference between a container and a virtual machine. Whereas, a virtual machine is full-blown operating system platform, running on a host OS, a container is not.

Containers are a remarkable technology that enable you to expand your Linode in ways you could not have done so otherwise. For example, you could deploy multiple instances of Nginx, even with multiple stagings (such as development and production). Unlike deploying multiple virtual machines, the deployed containers will not put nearly the hit on your Linode resources. Containers make this all very possible and Docker makes deploying containers very easy.

## Docker

Docker is only one technology capable of creating, deploying, and managing containers. It does, however, happen to be one of the technologies that makes working with containers quite simple. Let’s take a quick glance at some of the basics that make up the regular interaction with Docker.

### Images

The first piece of the puzzle you must understand is that of images. Every container you deploy is created from an image. You pull images from a Docker registry (such as the official [Docker Hub](https://hub.docker.com/)) and use them to build containers. A single image can create numerous containers. For example, you could use the latest [nginx image](https://hub.docker.com/_/nginx/) to deploy a web server container for:

* Web dev ops
* Testing
* Production
* Web applications

The possibilities are nearly endless.

### Dockerfiles

Dockerfiles are text files that contain the necessary commands to assemble an image. Once a Dockerfile is written, the administrator uses the *docker build* command to create the image based on the commands within the file. One very unique component of the Dockerfile is the ability to create an image based on your specifics. Say, for instance, you want to create an Ubuntu image to be used for a specific purpose, one that requires a fully updated version of Ubuntu and the inclusion of the tools for building software. With the help of a Dockerfile, you could make that happen with ease.

Each Dockerfile will make use of commands for building the images. The available commands are:

* ADD - copies the files from a source on the host into the container's own filesystem at the set destination
* CMD - can be used for executing a specific command within the container
* ENTRYPOINT - sets a default application to be used every time a container is created with the image
* ENV - sets environment variables
* EXPOSE - associates a specific port to enable networking between the container and the outside world
* FROM - defines the base image used to start the build process
* MAINTAINER - defines a full name and email address of the image creator
* RUN - central executing directive for Dockerfiles
* USER - sets the UID (or username) which is to run the container
* VOLUME - is used to enable access from the container to a directory on the host machine
* WORKDIR - sets the path where the command, defined with CMD, is to be executed

Not every command must be used. In fact, from the above list, you could craft a working Dockerfile by using the MAINTAINER, FROM, and RUN commands like so:

*MAINTAINER NAME EMAIL*

*FROM ubuntu:latest*

*RUN apt-get -y update && apt-get -y upgrade && apt-get install -y build-essential*

Where NAME and EMAIL are specific to to whomever is building the Dockerfile.

Dockerfiles make it incredibly easy to create the means by which you can quickly create multiple images, using the exact same specifications, with a single command.

### Docker Swarm

Docker makes it incredibly easy to join servers together to form a cluster, called a Docker Swarm. Once you’ve created a leader and joined nodes to the leader, you can easily scale out your container deployment. The Swarm manager (leader) will automatically adapt the cluster by adding or removing tasks to maintain a desired state. A node is a single instance of the Docker engine that participates in the Swarm. You can run one or more nodes on a single physical computer or a cloud server (such as a Linode). The Swarm manager uses ingress load balancing to expose services that are to be made available to the Swarm. Docker Swarm is also capable of:

* Checking the health of your containers
* Launching a fixed set of containers from a single Docker image
* Scaling the number of containers up or down (depending upon the current load)
* Perform rolling updates across containers
* Provide redundancy and failover
* Easily add or subtract container iterations as demands change

# To be continued

When next we visit this topic, we’ll start with the basics of installing Docker on a Linode, pulling an image, and running your first container. Until then, keep diving into the heart of Docker with the [official Docker documentation](https://docs.docker.com/).