How To Install and Use Docker: Getting Started

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=258

Docker

Whether it be from your development machine to a remote server for production, or packaging everything for use elsewhere, it is always a challenge when it comes to porting your application stack together with its dependencies and getting it to run without hiccups. In fact, the challenge is immense and solutions so far have not really proved successful for the masses.

In a nutshell, docker as a project offers you the complete set of higher-level tools to carry everything that forms an application across systems and machines - virtual or physical - and brings along loads more of great benefits with it.

Docker achieves its robust application (and therefore, process and resource) containment via Linux Containers (e.g. namespaces and other kernel features). Its further capabilities come from a project's own parts and components, which extract all the complexity of working with lower-level linux tools/APIs used for system and application management with regards to securely containing processes.

The Docker Project and its Main Parts

Docker project (open-sourced by dotCloud in March '13) consists of several main parts (applications) and elements (used by these parts) which are all [mostly] built on top of already existing functionality, libraries and frameworks offered by the Linux kernel and third-parties (e.g. LXC, device-mapper, aufs etc.).

Main Docker Parts

1. docker daemon: used to manage docker (LXC) containers on the host it runs
2. docker CLI: used to command and communicate with the docker daemon
3. docker image index: a repository (public or private) for docker images

Main Docker Elements

1. docker containers: directories containing everything-your-application
2. docker images: snapshots of containers or base OS (e.g. Ubuntu) images
3. Dockerfiles: scripts automating the building process of images

Docker Elements

The following elements are used by the applications forming the docker project.

Docker Containers

The entire procedure of porting applications using docker relies solely on the shipment of containers.

Docker containers are basically directories which can be packed (e.g. tar-archived) like any other, then shared and run across various different machines and platforms (hosts). The only dependency is having the hosts tuned to run the containers (i.e. have docker installed). Containment here is obtained via Linux Containers (LXC).

LXC (Linux Containers)

Linux Containers can be defined as a combination various kernel-level features (i.e. things that Linux-kernel can do) which allow management of applications (and resources they use) contained within their own environment. By making use of certain features (e.g. namespaces, chroots, cgroups and SELinux profiles), the LXC contains application processes and helps with their management through limiting resources, not allowing reach beyond their own file-system (access to the parent's namespace) etc.

Docker* with its *containers makes use of LXC, however, also brings along much more.

Docker Containers

Docker containers have several main features.

They allow;

- Application portability
- Isolating processes
- Prevention from tempering with the outside
- Managing resource consumption

and more, requiring much less resources than traditional virtual-machines used for isolated application deployments.

They do not allow;

- Messing with other processes
- Causing "dependency hell"
- Or not working on a different system
- Being vulnerable to attacks and abuse all system's resources

and (also) more.

Being based and depending on LXC, from a technical aspect, these containers are like a directory (but a shaped and formatted one). This allows portability and gradual builds of containers.

Each container is layered like an onion and each action taken within a container consists of putting another

block (which actually translates to a simple change within the file system) on top of the previous one. And various tools and configurations make this set-up work in a harmonious way altogether (e.g. union file-system).

What this way of having containers allows is the extreme benefit of easily launching and creating new containers and images, which are thus kept lightweight (thanks to gradual and layered way they are built). Since everything is based on the file-system, taking snapshots and performing roll-backs in time are cheap (i.e. very easily done / not heavy on resources), much like version control systems (VCS).

Each docker container starts from a docker image which forms the base for other applications and layers to come.

Docker Images

Docker images constitute the base of docker containers from which everything starts to form. They are very similar to default operating-system disk images which are used to run applications on servers or desktop computers.

Having these images (e.g. Ubuntu base) allow seamless portability across systems. They make a solid, consistent and dependable base with everything that is needed to run the applications. When everything is self-contained and the risk of system-level updates or modifications are eliminated, the container becomes immune to external exposures which could put it out of order -preventing the dependency hell.

As more layers (tools, applications etc.) are added on top of the base, new images can be formed by committing** these changes. When a new container gets created from a saved (i.e. committed) image, things continue from where they left off. And the **union file system, brings all the layers together as a single entity when you work with a container.

These base images can be explicitly stated when working with the docker CLI* to directly create a new container or they might be specified inside a *Dockerfile for automated image building.

Dockerfiles

Dockerfiles are scripts containing a successive series of instructions, directions, and commands which are to be executed to form a new docker image. Each command executed translates to a new layer of the onion, forming the end product. They basically replace the process of doing everything manually and repeatedly. When a Dockerfile is finished executing, you end up having formed an image, which then you use to start (i.e. create) a new container.

How To Install Docker

At first, docker was only available on Ubuntu. Nowadays, with its most recent release (0.7.1. dating 5 Dec.), it is possible to deploy docker on RHEL based systems (e.g. CentOS) and others as well.

Installation Instructions for Ubuntu

The simplest way to get docker, other than using the pre-built application image, is to go with a 64-bit Ubuntu 13.04 server

Update:

```
sudo aptitude    update
sudo aptitude -y upgrade
```

Make sure aufs support is available:

```
sudo aptitude install linux-image-extra-`uname -r`
```

Add docker repository key to apt-key for package verification:

```
sudo sh -c "wget -qO- https://get.docker.io/gpg | apt-key add -"
```

Add the docker repository to aptitude sources:

```
sudo sh -c "echo deb http://get.docker.io/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
```

Update the repository with the new addition:

```
sudo aptitude    update
```

Finally, download and install docker:

```
sudo aptitude install lxc-docker
```

Ubuntu's default firewall (UFW: Uncomplicated Firewall) denies all forwarding traffic by default, which is needed by docker.

Enable forwarding with UFW:

Edit UFW configuration using the nano text editor.

```
sudo nano /etc/default/ufw
```

Scroll down and find the line beginning with DEFAULT_FORWARD_POLICY.

Replace:

```
DEFAULT_FORWARD_POLICY="DROP"
```

With:

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Press CTRL+X and approve with Y to save and close.

Finally, reload the UFW:

```
sudo ufw reload
```

How To Use Docker

Once you have docker installed, its intuitive usage experience makes it very easy to work with. By now, you shall have the docker daemon running in the background. If not, use the following command to run the docker daemon.

To run the docker daemon:

```
sudo docker -d &
```

Usage Syntax:

Using docker (via CLI) consists of passing it a chain of options and commands followed by arguments. Please note that docker needs sudo privileges in order to work.

```
sudo docker [option] [command] [arguments]
```

Note: Below instructions and explanations are provided to be used as a guide and to give you an overall idea of using and working with docker. The best way to get familiar with it is practice on a new server. Do not be afraid of breaking anythingâ€“ in fact, do break things! With docker, you can save your progress and continue from there very easily.

Beginning

Let's begin with seeing all available commands docker have.

Ask docker for a list of all available commands:

```
sudo docker
```

All currently (as of 0.7.1) available commands:

```
attach    Attach to a running container
build     Build a container from a Dockerfile
commit    Create a new image from a container's changes
cp        Copy files/folders from the containers filesystem to the host path
diff      Inspect changes on a container's filesystem
events    Get real time events from the server
export    Stream the contents of a container as a tar archive
history   Show the history of an image
images    List images
import    Create a new filesystem image from the contents of a tarball
info      Display system-wide information
insert    Insert a file in an image
inspect   Return low-level information on a container
```

```
kill      Kill a running container
load      Load an image from a tar archive
login     Register or Login to the docker registry server
logs      Fetch the logs of a container
port      Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
ps        List containers
pull      Pull an image or a repository from the docker registry server
push      Push an image or a repository to the docker registry server
restart   Restart a running container
rm        Remove one or more containers
rmi       Remove one or more images
run       Run a command in a new container
save      Save an image to a tar archive
search    Search for an image in the docker index
start     Start a stopped container
stop      Stop a running container
tag       Tag an image into a repository
top       Lookup the running processes of a container
version   Show the docker version information
wait      Block until a container stops, then print its exit code
```

Check out system-wide information and docker version:

```
# For system-wide information on docker:
sudo docker info
# For docker version:
sudo docker version
```

Working with Images

As we have discussed at length, the key to start working with any docker container is using images. There are many freely available images shared across docker image index and the CLI allows simple access to query the image repository and to download new ones.

When you are ready, you can also share your image there as well. See the section on "pushâ€• further down for details.

Searching for a docker image:*

```
# Usage: sudo docker search [image name]
sudo docker search ubuntu
```

This will provide you avery long*list of all available images matching the query:* ***Ubuntu.***

Downloading (PULLing) an image:

Either when you are building / creating a container or before you do, you will need to have an image present at the host machine where the containers will exist. In order to download images (perhaps following "searchâ€•) you can execute pull to get one.

```
# Usage: sudo docker pull [image name]
sudo docker pull ubuntu
```

Listing images:

All the images on your system, including the ones you have created by committing (see below for details), can be listed using "imagesâ€•. This provides a full list of all available ones.

```
# Example: sudo docker images
sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
my_img              latest              72461793563e        36 seconds ago
128 MB
ubuntu              12.04               8dbd9e392a96        8 months ago
128 MB
ubuntu              latest              8dbd9e392a96        8 months ago
128 MB
ubuntu              precise             8dbd9e392a96        8 months ago
128 MB
ubuntu              12.10               b750fe79269d        8 months ago
175.3 MB
ubuntu              quantal             b750fe79269d        8 months ago
175.3 MB
```

Committing changes to an image:

As you work with a container and continue to perform actions on it (e.g. download and install software, configure files etc.), to have it keep its state, you need to "commitâ€•. Committing makes sure that everything continues from where they left next time you use one (i.e. an image).

```
# Usage: sudo docker commit [container ID] [image name]
sudo docker commit 8dbd9e392a96 my_img
```

Sharing (PUSHing) images:

Although it is a bit early at this moment -in our article*, when you have created your own container which you would like to share with the rest of the world, you can use push to have* ***your image listed in the index where everybody can download and use.***

Please remember to "commitâ€• all your changes.

```
# Usage: sudo docker push [username/image name]
sudo docker push my_username/my_first_image
```

Note:You need to sign-up at index.docker.io to push images to docker index.

Working with Containers

When you "run" any process using an image, in return, you will have a container. When the process is not* actively running, this container will be a *non-running container. Nonetheless, all of them will reside on your system until you remove them via rm command.

Listing all current containers:

By default, you can use the following to list all running containers:

```
sudo docker ps
```

To have a list of both running and non-running ones, use:

```
sudo docker ps -l
```

Creating a New Container

It is currently not possible to create a container without running anything (i.e. commands). To create a new container, you need to use a base image and specify a command to run.

```
# Usage: sudo docker run [image name] [command to run]
sudo docker run my_img echo "hello"
# To name a container instead of having long IDs
# Usage: sudo docker run -name [name] [image name] [comm.]
sudo docker run my_img -name my_cont_1 echo "hello"
```

This will output "hello" and you will be right back where you were. (i.e. your host's shell)

As you can not change the command you run after having created a container (hence specifying one during "creation"), it is common practice to use process managers and even custom launch scripts to be able to execute different commands.

Running a container:

When you create a container and it stops (either due to its process ending or you stopping it explicitly), you can use "runâ€• to get the container working again with the same command used to create it.

```
# Usage: sudo docker run [container ID]
sudo docker run c629b7d70666
```

Remember how to find the containers? See above section for *list*ing them.

Stopping a container:

To stop a container's process from running:

```
# Usage: sudo docker stop [container ID]
sudo docker stop c629b7d70666
```

Saving (*commit*ting) a container:

If you would like to save the progress and changes you made with a container, you can use "commitâ€• as explained above to save it as an image.

This command turns your container to an image.

Remember that with docker, commits are cheap. Do not hesitate to use them to create images to save your progress with a
container or to roll back when you need (e.g. like snapshots in time).

Removing / Deleting a container:

Using the ID of a container, you can delete one with rm.

```
# Usage: sudo docker rm [container ID]
sudo docker rm c629b7d70666
```