**[Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context](http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically.php)**

**Introduction**

We create Docker containers using **[base]** images. An image can be basic, with nothing but the operating-system fundamentals, or it can consist of a sophisticated pre-built application stack ready for launch.

When we build images with docker, each action taken (i.e. a command executed such as **apt-get install**) forms a new layer on top of the previous one. These base images then can be used to create new containers.

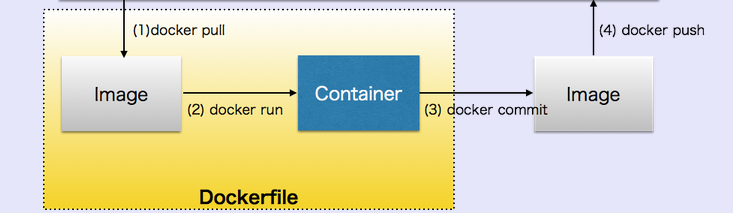


Image source: [Docker](http://hondou.homedns.org/pukiwiki/pukiwiki.php?Docker%20%A4%B6%A4%C3%A4%AF%A4%EA%B8%C0%A4%A6%A4%C8%B2%BF%3F" \t "_blank)

In this chapter, we're going to learn how to automate this process via instructions in **Dockerfiles**. A Dockerfile is a text document that contains all the commands we would normally execute manually in order to build a Docker image. By calling docker build from our terminal, we can have Docker build our image executing the instructions successively step-by-step, layer-by-layer, automatically from a source (base) image.

The docker project offers higher-level tools which work together, built on top of some Linux kernel features. The goal is to help developers and system administrators port applications with all of their dependencies included, and get them running across systems and machines headache free.

Docker achieves this by creating safe, LXC (Linux Containers) based environments for applications called docker containers. These containers are created using docker images, which can be built either by executing commands manually or automatically through Dockerfiles.

**Dockerfile**

Each Dockerfile is a script, composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image in order to create (or form) a new one. They are used for organizing things and greatly help with deployments by simplifying the process start-to-finish.

Dockerfiles begin with defining an image FROM which the build process starts. Followed by various other methods, commands and arguments (or conditions), in return, provide a new image which is to be used for creating docker containers.

**Sample Dockerfile**

Here is our Dockerfile we're going to playing with in this chapter. We'll run instructions from this file step by step by uncommenting and commenting each line.

FROM debian:latest

MAINTAINER devops@bogotobogo.com

# 1 - RUN

RUN apt-get update && DEBIAN\_FRONTEND=noninteractive apt-get install -yq apt-utils

RUN DEBIAN\_FRONTEND=noninteractive apt-get install -yq htop

RUN apt-get clean

# 2 - CMD

CMD ["htop"]

CMD ["ls", "-l"]

# 3 - WORKDIR and ENV

WORKDIR /root

ENV DZ version1

# 4 - ADD

ADD run.sh /root/run.sh

CMD ["./run.sh"]

# 5 - ENTRYPOINT (vs CMD)

ENTRYPOINT ["./run.sh"]

CMD ["arg1"]

**Dockerfile format**

Here is the format of the Dockerfile:

# Comment

INSTRUCTION arguments

The Instruction is not case-sensitive, however convention is for them to be UPPERCASE in order to distinguish them from arguments more easily.

Docker runs the instructions in a Dockerfile in order. The first instruction must be FROM in order to specify the Base Image from which we are building.

Docker will treat lines that begin with # as a comment.

**FROM**

The FROM instruction sets the Base Image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. The image can be any valid image. But it is especially easier to start by pulling an image from the [Public Repositories](http://docs.docker.com/userguide/dockerrepos/#using-public-repositories).

**MAINTAINER**

MAINTAINER <name>

The MAINTAINER instruction allows us to set the Author field of the generated images.

**Docker build & context**

To build an image from a source repository, we create a description file called Dockerfile at the root of our repository. This file will describe the steps to assemble the image.

Then we call docker build with the path of our source repository as the argument:

Usage: docker build [OPTIONS] PATH | URL | -

It will build a new image from the source code at PATH.

Here is an example using the current directory (".") as its path:

$ sudo docker build .

The path to the source repository defines where to find the context of the build. The build is run by the Docker daemon, not by the CLI, so the whole context must be transferred to the daemon. The Docker CLI reports "Sending build context to Docker daemon" when the context is sent to the daemon.

Note: Avoid using root directory, /, as the root of the source repository. The docker build command will use whatever directory contains the Dockerfile as the build context (including all of its subdirectories). The build context will be sent to the Docker daemon before building the image, which means if we use / as the source repository, the entire contents of our hard drive will get sent to the daemon (and thus to the machine running the daemon). We probably don't want that. In most cases, it's best to put each Dockerfile in an empty directory, and then add only the files needed for building that Dockerfile to that directory. To further speed up the build, we can exclude files and directories by adding a .dockerignore file to the same directory.

The Docker daemon will run our steps one-by-one, committing the result to a new image if necessary, before finally outputting the ID of our new image. The Docker daemon will automatically clean up the context we sent.

Note that each instruction is run independently, and causes a new image to be created.

Whenever possible, Docker will re-use the intermediate images, accelerating docker build significantly.

$ docker build --help

Usage: docker build [OPTIONS] PATH | URL | -

Build a new image from the source code at PATH

-t, --tag="" Repository name (and optionally a tag) to be applied to the resulting image in case of success