[Dockerfile - Build Docker images automatically II - revisiting FROM, MAINTAINER, build context, and caching](http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically_2.php)

Continued from ...

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

This chapter is similar to the previous one, [Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context](http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically.php), that's because we want to make sure how **docker build** works with **context**.

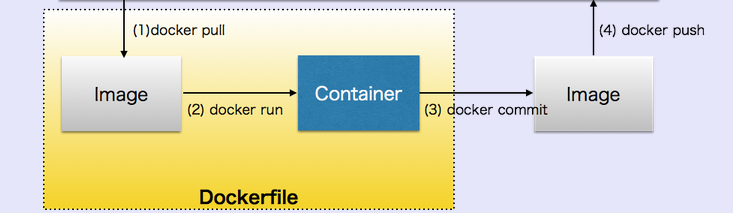


Image source: [Docker](http://hondou.homedns.org/pukiwiki/pukiwiki.php?Docker%20%A4%B6%A4%C3%A4%AF%A4%EA%B8%C0%A4%A6%A4%C8%B2%BF%3F)

In this chapter, we're going to learn more on how to automate this process via instructions in Dockerfiles.

Base image

Let's download our base image:

$ docker pull debian:latest

debian:latest: The image you are pulling has been verified

511136ea3c5a: Pull complete

f10807909bc5: Pull complete

f6fab3b798be: Pull complete

Status: Downloaded newer image for debian:latest

k@laptop:~/Documents/demo$ docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

debian latest f6fab3b798be 2 weeks ago 85.1 MB

In our local working directory, we have only one file, Dockerfile:

k@laptop:~/Documents/demo$ ls

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.

docker build 'FROM'

Let's look at the syntax of docker build command:

$ 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

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.

FROM debian:latest

MAINTAINER devops@bogotobogo.com

Let's run docker build command with the two-line Dockerfile:

k@laptop:~/Documents/demo$ docker build -t bogodevops/demo:v1 .

Sending build context to Docker daemon 2.56 kB

Sending build context to Docker daemon

Step 0 : FROM debian:latest

---> f6fab3b798be

Step 1 : MAINTAINER k@bogotobogo.com

---> Running in 4181b54ab22e

---> 511bcbdd59ba

Removing intermediate container 4181b54ab22e

Successfully built 511bcbdd59ba

Now if we list the images:

k@laptop:~/Documents/demo$ docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

bogodevops/demo v1 511bcbdd59ba About a minute ago 85.1 MB

debian latest f6fab3b798be 2 weeks ago 85.1 MB

Note that 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 (2.56kB) is sent to the daemon as shown in the output:

Sending build context to Docker daemon 2.56 kB

If we send big chuck to the daemon, it will take longer to copy things. For example, if we send duplicate device files(/de/zero) with dd:

k@laptop:~/Documents/demo$ dd if=/dev/zero of=testimage bs=4096 count=8192

8192+0 records in

8192+0 records out

33554432 bytes (34 MB) copied, 0.118561 s, 283 MB/s

k@laptop:~/Documents/demo$ ls

Dockerfile testimage

k@laptop:~/Documents/demo$ docker build -t bogodevops/demo:v1 .

Sending build context to Docker daemon 33.56 MB

Sending build context to Docker daemon

Step 0 : FROM debian:latest

---> f6fab3b798be

Step 1 : MAINTAINER k@bogotobogo.com

---> Using cache

---> 511bcbdd59ba

Successfully built 511bcbdd59ba

Note that the size has been increased from 2.56kb to 33.56MB. That's why the [Docker document](http://docs.docker.com/reference/builder/) gives us a Warning like this:

"Warning Avoid using your root directory, /, as the root of the source repository. The docker build command will use whatever dicrectory 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 you use / as the source repository, the entire contents of your hard drive will get sent to the daemon (and thus to the machine running the daemon). You probably don't want that."

Or like this:

"Warning: Do not use your root directory, /, as the PATH as it causes the build to transfer the entire contents of your hard drive to the Docker daemon.

So, we should be aware of the context of our build directory!

Again:

"The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it's best to start with an empty directory as **context** and keep your **Dockerfile** in that directory. Add only the files needed for building the **Dockerfile**"

Also note the the difference in the Step 1 of the two cases:

In the first instance of docker build:

Step 1 : MAINTAINER k@bogotobogo.com

---> Running in 4181b54ab22e

---> 511bcbdd59ba

But in the second run, Docker used cache:

Step 1 : MAINTAINER k@bogotobogo.com

---> Using cache

---> 511bcbdd59ba

When we build a Docker image, it's using a Dockerfile, and every instruction in the Dockerfile is run inside of a container. If that returns successfully, then that container is stored as a new image.

In our case, in Step 0, we created 'f6fab3b798be' which is a hash identifier, and Step 1, we created '511bcbdd59ba' hash.

Note that in our 2nd run (the 'docker run' with 'dd'), the hash is the same. What does this mean? If the instructions in our Dockerfile are the same, Docker uses the cache:

k@laptop:~/Documents/demo$ docker images -a

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

bogodevops/demo v1 511bcbdd59ba 57 minutes ago 85.1 MB

debian latest f6fab3b798be 2 weeks ago 85.1 MB

<none> <none> f10807909bc5 2 weeks ago 85.1 MB

<none> <none> 511136ea3c5a 17 months ago 0 B

So, every step along the way, we create a new image. As it succeeds, we'll build a new layer on top of the previous one as we read in an instruction. As this caching allows us to build other environment similar to the previous image without rebuilding from every steps involved.