[Dockerfile - Build Docker images automatically IV - CMD](http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically_4_CMD.php)

Continued from ...

Continued from [Dockerfile - Build Docker images automatically III - RUN](http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically_3.php)

In this chapter, we're going to learn more on how to automate this process via instructions in Dockerfiles. We'll be focused on **CMD**.

Dockerfie - CMD

This section is from <http://docs.docker.com/reference/builder/>.

CMD has 3 forms:

1. CMD ["executable","param1","param2"] (exec form, this is the preferred form)
2. CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
3. CMD command param1 param2 (shell form)

There can only be one CMD instruction in a Dockerfile. If we list more than one CMD then only the last CMD will take effect.

The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case we must specify an ENTRYPOINT instruction as well.

1. **Note**: If CMD is used to provide default arguments for the ENTRYPOINT instruction, both the CMD and ENTRYPOINT instructions should be specified with the JSON array format.
2. **Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').
3. **Note**: Unlike the shell form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, CMD [ "echo", "$HOME" ] will not do variable substitution on $HOME. If you want shell processing then either use the shell form or execute a shell directly, for example: CMD [ "sh", "-c", "echo", "$HOME" ].

Dockerfile 'CMD' sample

Here is our Dockerfile that we're going to play 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"]

We have one instruction for CMD, and at the completion of the CMD, it will become an image.

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

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

Step 2 : RUN apt-get update && DEBIAN\_FRONTEND=noninteractive apt-get install -yq apt-utils

---> Using cache

---> e6e2c03b8efc

Step 3 : RUN DEBIAN\_FRONTEND=noninteractive apt-get install -yq htop

---> Using cache

---> fac6e3168cfe

Step 4 : RUN apt-get clean

---> Using cache

---> 358b5cc4b9fa

Step 5 : CMD htop

---> Running in d31a73253846

---> b64547129d16

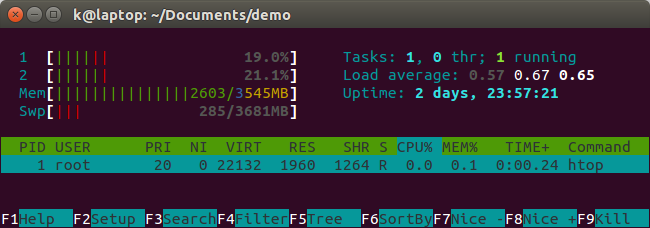
Removing intermediate container d31a73253846

Successfully built b64547129d16

But unlike in the previous chapter where we ran htop explicitly within the container, this time, it becomes a default environment.

So, even though we issue docker run without passing in any command, we have htop running automatically when the container is created:

k@laptop:~/Documents/demo$ docker run -it --rm bogodevops/demo



We get the htop as soon as we're in the container. It's given us as an environment.

If we pass in /bin/bash, then we'll have bash instead of htop:

k@laptop:~/Documents/demo$ docker run -it --rm bogodevops/demo /bin/bash

root@00e40007ed7d:/# exit

exit

Before we start new thing, we need to remove 'testimage' in our directory:

k@laptop:~/Documents/demo$ ls

Dockerfile testimage

k@laptop:~/Documents/demo$ rm testimage

Then, let's switch our CMD instruction to CMD ["ls", "l"]. Here is our new Dockerfile:

FROM debian:latest

MAINTAINER k@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"]

Build a new image with the new CMD ["ls", "l"]:

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

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

---> Using cache

---> 511bcbdd59ba

Step 2 : RUN apt-get update && DEBIAN\_FRONTEND=noninteractive apt-get install -yq apt-utils

---> Using cache

---> e6e2c03b8efc

Step 3 : RUN DEBIAN\_FRONTEND=noninteractive apt-get install -yq htop

---> Using cache

---> fac6e3168cfe

Step 4 : RUN apt-get clean

---> Using cache

---> 358b5cc4b9fa

Step 5 : CMD ls -l

---> Running in 717df1a3baa2

---> d2f3de97b6ef

Removing intermediate container 717df1a3baa2

Successfully built d2f3de97b6ef

If we go in our container, it will automatically gives the output from ls -a:

k@laptop:~/Documents/demo$ docker run -it --rm bogodevops/demo

total 68

drwxr-xr-x 2 root root 4096 Nov 5 21:37 bin

drwxr-xr-x 2 root root 4096 Sep 21 18:17 boot

drwxr-xr-x 4 root root 360 Nov 25 05:25 dev

drwxr-xr-x 32 root root 4096 Nov 25 05:25 etc

drwxr-xr-x 2 root root 4096 Sep 21 18:17 home

drwxr-xr-x 8 root root 4096 Nov 25 02:27 lib

drwxr-xr-x 2 root root 4096 Nov 5 21:33 lib64

drwxr-xr-x 2 root root 4096 Nov 5 21:31 media

drwxr-xr-x 2 root root 4096 Sep 21 18:17 mnt

drwxr-xr-x 2 root root 4096 Nov 5 21:31 opt

dr-xr-xr-x 253 root root 0 Nov 25 05:25 proc

drwx------ 2 root root 4096 Nov 5 21:31 root

drwxr-xr-x 5 root root 4096 Nov 5 21:37 run

drwxr-xr-x 2 root root 4096 Nov 5 21:37 sbin

drwxr-xr-x 2 root root 4096 Jun 10 2012 selinux

drwxr-xr-x 2 root root 4096 Nov 5 21:31 srv

dr-xr-xr-x 13 root root 0 Nov 25 05:25 sys

drwxrwxrwt 2 root root 4096 Nov 5 21:37 tmp

drwxr-xr-x 16 root root 4096 Nov 25 02:27 usr

drwxr-xr-x 17 root root 4096 Nov 25 02:27 var