Yury Pitsishin

on Software Design, Big Data and DevOps

April 2, 2016

Docker RUN vs CMD vs ENTRYPOINT

Some Docker instructions look similar and cause confusion among developers who just started using Docker or do it irregularly. In this post I will explain the difference between CMD, RUN, and ENTRYPOINT on examples.

In a nutshell

- RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.
- CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.
- ENTRYPOINT configures a container that will run as an executable.

If it doesn’t make much sense or you after details, then read on.

Docker images and layers

When Docker runs a container, it runs an *image* inside it. This image is usually built by executing Docker instructions, which add *layers* on top of existing image or *OS distribution*. *OS distribution* is the initial image and every added layer creates a new image.

Final Docker *image* reminds an onion with OS distribution inside and a number of layers on top of it. For example, your image can be built by installing a number of deb packages and your application on top of Ubuntu 14.04 distribution.

Shell and Exec forms

All three instructions (RUN, CMD and ENTRYPOINT) can be specified in *shell* form or *exec* form. Let's get familiar with these forms first, because the forms usually cause more confusion than instructions themselves.

Shell form

`<instruction> <command>`

Examples:

```
RUN apt-get install python3
CMD echo "Hello world"
ENTRYPOINT echo "Hello world"
```

When instruction is executed in *shell* form it calls `/bin/sh -c <command>` under the hood and normal shell processing happens. For example, the following snippet in Dockerfile

```
ENV name John Dow
ENTRYPOINT echo "Hello, $name"
```

when container runs as `docker run -it <image>` will produce output

```
Hello, John Dow
```

Note that variable *name* is replaced with its value.

Exec form

This is the preferred form for CMD and ENTRYPOINT instructions.

```
<instruction> ["executable", "param1", "param2", ...]
```

Examples:

```
RUN ["apt-get", "install", "python3"]
CMD ["/bin/echo", "Hello world"]
ENTRYPOINT ["/bin/echo", "Hello world"]
```

When instruction is executed in *exec* form it calls executable directly, and shell processing does not happen. For example, the following snippet in Dockerfile

```
ENV name John Dow
ENTRYPOINT ["/bin/echo", "Hello, $name"]
```

when container runs as `docker run -it <image>` will produce output

```
Hello, $name
```

Note that variable *name* is not substituted.

How to run bash?

If you need to run *bash* (or any other interpreter but sh), use *exec* form with `/bin/bash` as executable. In this case, normal shell processing will take place. For example, the following snippet in Dockerfile

```
ENV name John Dow
ENTRYPOINT ["/bin/bash", "-c", "echo Hello, $name"]
```

when container runs as `docker run -it <image>` will produce output

```
Hello, John Dow
```

RUN

RUN instruction allows you to install your application and packages requited for it. It executes any commands on top of the current image and creates a new layer by committing the results. Often you will find multiple RUN instructions in a Dockerfile.

RUN has two forms:

- `RUN <command>` (shell form)
- `RUN ["executable", "param1", "param2"]` (exec form)

(The forms are described in detail in *Shell and Exec forms* section above.)

A good illustration of RUN instruction would be to install multiple version control systems packages:

```
RUN apt-get update && apt-get install -y \
  bzr \
  cvs \
  git \
  mercurial \
  subversion
```

Note that `apt-get update` and `apt-get install` are executed in a single RUN instruction. This is done to make sure that the latest packages will be installed. If `apt-get install` were in a separate RUN instruction, then it would reuse a layer added by `apt-get update`, which could had been created a long time ago.

CMD

CMD instruction allows you to set a *default* command, which will be executed only when you run container without specifying a command. If Docker container runs with a command, the default command will be ignored. If Dockerfile has more than one CMD instruction, all but last CMD instructions are ignored.

CMD has three forms:

- `CMD ["executable","param1","param2"]` (exec form, preferred)
- `CMD ["param1","param2"]` (sets additional default parameters for ENTRYPOINT in *exec*

form)

- `CMD command param1 param2` (shell form)

Again, the first and third forms were explained in *Shell and Exec forms* section. The second one is used together with ENTRYPOINT instruction in *exec* form. It sets default parameters that will be added after ENTRYPOINT parameters if container runs without command line arguments. See ENTRYPOINT for example.

Let’s have a look how CMD instruction works. The following snippet in Dockerfile

```
CMD echo "Hello world"
```

when container runs as `docker run -it <image>` will produce output

```
Hello world
```

but when container runs with a command, e.g., `docker run -it <image> /bin/bash`, CMD is ignored and bash interpreter runs instead:

```
root@7de4bed89922:/#
```

ENTRYPOINT

ENTRYPOINT instruction allows you to configure a container that will run as an executable. It looks similar to CMD, because it also allows you to specify a command with parameters. The difference is ENTRYPOINT command and parameters are not ignored when Docker container runs with command line parameters. (There is a way to ignore ENTTRYPOINT, but it is unlikely that you will do it.)

ENTRYPOINT has two forms:

- `ENTRYPOINT ["executable", "param1", "param2"]` (exec form, preferred)
- `ENTRYPOINT command param1 param2` (shell form)

Be very careful when choosing ENTRYPOINT form, because forms behaviour differs significantly.

Exec form

Exec form of ENTRYPOINT allows you to set commands and parameters and then use either form of CMD to set additional parameters that are more likely to be changed. ENTRYPOINT arguments are always used, while CMD ones can be overwritten by command line arguments provided when Docker container runs. For example, the following snippet in Dockerfile

```
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["world"]
```

when container runs as `docker run -it <image>` will produce output

```
Hello world
```

but when container runs as `docker run -it <image> John` will result in

```
Hello John
```

Shell form

Shell form of ENTRYPOINT ignores any CMD or docker run command line arguments.

The bottom line

Use RUN instructions to build your image by adding layers on top of initial image.

Prefer ENTRYPOINT to CMD when building executable Docker image and you need a command always to be executed. Additionally use CMD if you need to provide extra default arguments that could be overwritten from command line when docker container runs.

Choose CMD if you need to provide a default command and/or arguments that can be overwritten from command line when docker container runs.

ALSO ON **GO IN BIG DATA**

4 years ago • 3 comments
Example Of Using Templates In Golang

4 years ago • 1 comment
Testing Go Code With Testify

4 years
Dem ifcol netw

Comments **Community** **Privacy Policy** 1 **Login**

Recommend 208 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Nitin Bansal • 2 years ago
Wow!!! So beautifully explained.....thanks so much :)
34 | • Reply • Share ›

Haritha Srinivas • a year ago
Such a great teacher you are .Awesome explanation in a simple way
26 | • Reply • Share ›

Serguei Cambour • 2 years ago
Reaally great explained nuances I've never seen before ! Thanks a lot, спасибо огромное, Юра !
8 | • Reply • Share ›

Pepito Fernandez • a year ago
Yury! Killed it. Added it to my bookmarks bar... then I realized... I won't need it.
6 | • Reply • Share ›