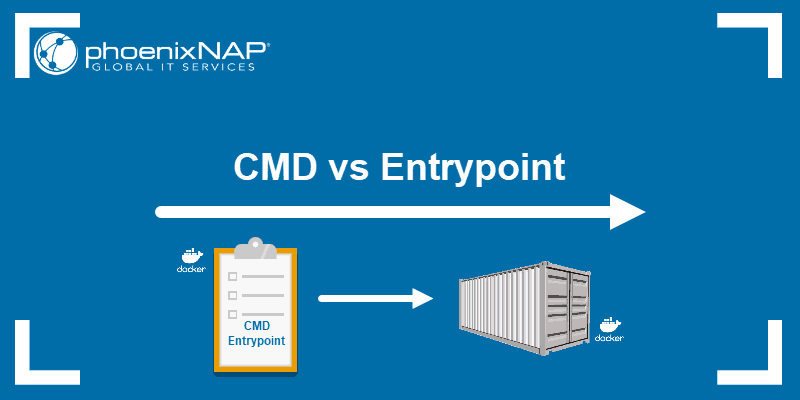
**ENTRYPOINT VS CMD**

Containers are **designed for running specific tasks and processes**, not for hosting operating systems. You create a container to serve a single unit task. **Once it completes the given task, it stops**. Therefore, the container life-cycle depends on the ongoing process inside of it. Once the process stops, the container stops as well.

A Dockerfile defines this process. It is a script made up of instructions on how to build a Docker image. In this script, there are **two types of instructions** that can define the process running in the container:

* **ENTRYPOINT**
* **CMD**



**Docker Entrypoint vs CMD: Solving the Dilemma**

In short, **CMD**defines default commands and/or parameters for a container. CMD is an instruction that is best to use if you need a default command which users can easily override. If a Dockerfile has multiple CMDs, it only applies the instructions from the last one.

On the other hand, **ENTRYPOINT** is preferred when you want to define a container with a specific executable. You cannot [override an ENTRYPOINT](https://phoenixnap.com/kb/docker-run-override-entrypoint) when starting a container unless you add the **--entrypoint** flag.

**Combine ENTRYPOINT with CMD** if you need a container with a specified executable and a default parameter that can be modified easily. For example, when [containerizing an application](https://phoenixnap.com/kb/how-to-containerize-applications) use **ENTRYPOINT** and **CMD** to set environment-specific variables.

## Shell and Exec Form

Before we begin, it is important to discus the forms of instructions. Docker ENTRYPOINT and CMD can have two forms:

* **Shell form**
* **Exec form**

The syntax for any command in shell form is:

<instruction> <command>

The syntax for instructions in exec form is:

<instruction> [“executable”, “parameter”]

You can write Docker CMD/ENTRYPOINT instructions in both forms:

* **CMD echo “Hello World”** (shell form)
* **CMD ["echo", "Hello World"]** (exec form)



* **ENTRYPOINT echo "Hello World"** (shell form)
* **ENTRYPOINT ["echo", "Hello World"]** (exec form)

## Docker CMD

**Docker CMD** defines the default executable of a Docker image. You can run this image as the base of a container without adding command-line arguments. In that case, the container runs the process specified by the CMD command.

The CMD instruction is only utilized if there is no argument added to the **run** command when starting a container. Therefore, if you add an argument to the command, you override the CMD.

### Creating a Dockerfile with CMD and Building an Image

1. Start by creating a new **MyDockerImage** folder to store your images in:

sudo mkdir MyDockerImage

2. Move into that folder and create a new Dockerfile:

cd MyDockerImage

sudo touch Dockerfile

3. Open the Dockerfile with your favorite text editor:

vi Dockerfile

4. Then, add the following content to the file:

FROM ubuntu

MAINTAINER sofija

RUN apt-get update

CMD [“echo”, “Hello World”]

To see CMD in action, we’ll create a container based on the image made in the previous step.  
Run the container with the command:

sudo docker run [image\_name]

Starting a container to test Docker CMD instruction.

Since there is no command-line argument, the container will run the default CMD instruction and display the **Hello World** message. However, if you add an argument when starting a container, it overrides the CMD instruction.

For example, add the hostname argument to the [docker run command](https://phoenixnap.com/kb/docker-run-command-with-examples):

sudo docker run [image\_name] hostname

Docker will run the container and the **hostname** command instead of the CMD’s echo command. You can see this in the output.

Example of how to override Docker CMD when starting a container.

## Docker Entrypoint

ENTRYPOINT is the other instruction used to configure how the container will run. Just like with CMD, you need to specify a command and parameters.

**What is the difference between CMD and ENTRYPOINT?** You cannot override the ENTRYPOINT instruction by adding command-line parameters to the **docker run** command. By opting for this instruction, you imply that the container is specifically built for such use.

Read on to see how we apply ENTRYPOINT in container creation.

### Creating a Dockerfile with ENTRYPOINT and Building an Image

1. Use the Dockerfile created in the CMD section and edit the file to change the instruction. Open the existing file with a text editor:

sudo vi Dockerfile

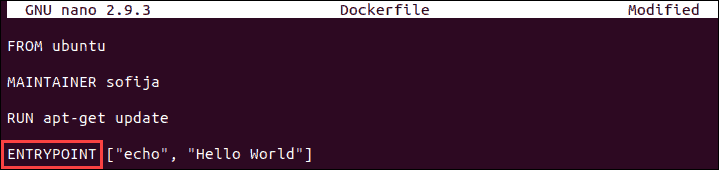
2. Edit the content by replacing the CMD command with ENTRYPOINT:

FROM ubuntu

MAINTAINER sofija

RUN apt-get update

ENTRYPOINT [“echo”, “Hello World”]



3. **Save** and **close** the file.

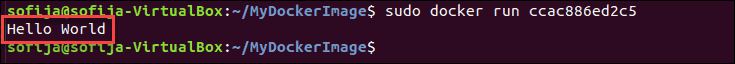
### Running a Docker Container with ENTRYPOINT

1. Build a new image using the following command:

sudo docker build .

2. The output should show you have successfully built the new image under a given name. Now let’s run it as a container without adding any command-line parameters:

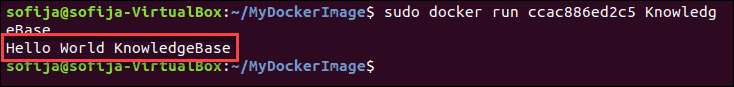
sudo docker run [image\_name]



The output will be the same as with CMD. This is because we haven’t added any arguments to the run command.

3. To see how ENTRYPOINT works, you need to add a parameter when starting a container. Use the same command as in the previous step and add something after the container name:

sudo docker run [image\_name] KnowledgeBase



As you see, Docker did not override the initial instruction of echoing Hello World. It merely added the new parameter to the existing command.

## Docker Entrypoint with CMD

As you have seen so far, ENTRYPOINT and CMD are similar, but not the same. What’s more, these two instructions are not mutually exclusive. That’s right, it is possible to have both in your Dockerfile.

There are many situations in which combining CMD and ENTRYPOINT would be the best solution for your Docker container. In such cases, the **executable is defined with ENTRYPOINT**, while **CMD specifies the default parameter**.

If you are using both instructions, make sure to keep them **in exec form**.

Read on to see how ENTRYPOINT and CMD collaborate in our example.

### Run a Container with Entrypoint and CMD

1. First, we are going to modify our existing Dockerfile so it includes both instructions. Open the file with:

sudo vi Dockerfile

2. The file should include an ENTRYPOINT instruction specifying the executable, as well as a CMD instruction defining the default parameter which should appear if no additional ones are added to the run command:

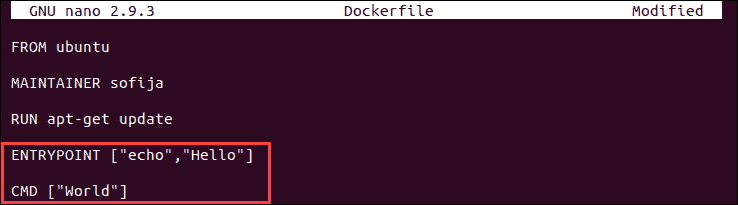
FROM ubuntu

MAINTAINER sofija

RUN apt-get update

ENTRYPOINT [“echo”, “Hello”]

CMD [“World”]



3. Now, build a new image from the modified Dockerfile:

sudo docker build .

4. Let’s test the container by running it without any parameters. Enter the command:

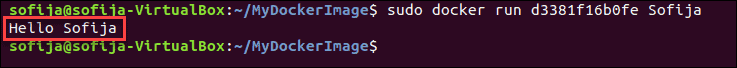
sudo docker run [image\_name]

Docker ENTRYPOINT vs CMD instructions combined.

It will return the message **Hello World**. However, what happens when we add parameters to the docker run command?

5. Use the same command again, but this time add your name to the run command:

sudo docker run [image\_name] [your\_name]



The output has now changed to **Hello [your\_name]**(in my case, it’s **Hello Sofija**). This is because you cannot override ENTRYPOINT instructions, whereas with CMD you can easily do so.