

# Docker: cmd VS entrypoint



Navee  Sep 28 · 4 min read

[#docker](#) [#dockerinstructions](#) [#cmd](#) [#entrypoint](#)

Docker is widely used these days, but if you have not got any chance to work on it, then its high time to get familiar with it. This post is not going to be a tutorial about docker though, so it assumes that you have its basic understanding. If you know docker, you must know that to build a docker image, you need to create Dockerfile with several instructions in it. In this post, I am going to talk about two of those instructions: `CMD` and `ENTRYPOINT` which can be a bit confusing if you haven't paid much attention to it.

Both of these docker instructions are used to define command/executables which is executed during container invocation. These are very similar but different instructions which can be used independently or together to achieve better flexibility to define what a container should execute. There are two ways/syntaxes to define them:

Exec form: (Preferred form)

```
CMD ["executable", "arg1", "arg2"]
```

```
ENTRYPOINT ["executable", "arg1", "arg2"]
```

Shell form:

```
CMD executable arg1 arg2  
ENTRYPOINT executable arg1 arg2
```

Use cases:

## CMD

Used to provide all the default scenarios which can be overridden.

Cmd instruction is used to define:

### **Default executable**

This instructions is used to define a default executable for a container to execute. If you want to create a generic docker image, where users can pass any supported command to be executed on container invocation, then this instruction is the one to use. Entrypoint instruction should not be defined in Dockerfile for this use case.

```
CMD ["executable", "arg1", "arg2"]
```

### **Default arguments**

This instruction can also be used to provide default arguments. For this use case, we don't specify executable in this instruction at all, but simply define some arguments which are used as default/additional arguments for executable defined in the entrypoint instruction. Thus, entrypoint

instruction is required in dockerfile for this use case to define an executable.

```
CMD ["arg1", "arg2"]
```

```
ENTRYPOINT ["executable"]
```

P.S: Anything defined in CMD can be overridden by passing arguments in `docker run` command.

## ENTRYPOINT

Used to define specific executable and arguments to be executed during container invocation which cannot be overridden. This is used to constraint the user to execute anything else. User can however define arguments to be passed in the executable by adding them in the docker run command.

```
ENTRYPOINT ["executable", "arg1", "arg2"]
```

## Demo

For the demo, let's take the following dockerFile which fetches necessary cowsay and screenfetch libraries so that we can run these commands when running the container.

### Demo 1 - Use of CMD to define Default executable

```
FROM debian:jessie-slim
```

```
RUN apt-get update                                && \
    apt-get install -y --no-install-recommends    \
        cowsay                                    \
        screenfetch                               && \
    rm -rf /var/lib/apt/lists/*

ENV PATH "$PATH:/usr/games"
CMD ["cowsay", "Yo, CMD !!"]
```

## Build the Image:

```
docker build -t demo .
```

## Run the container:

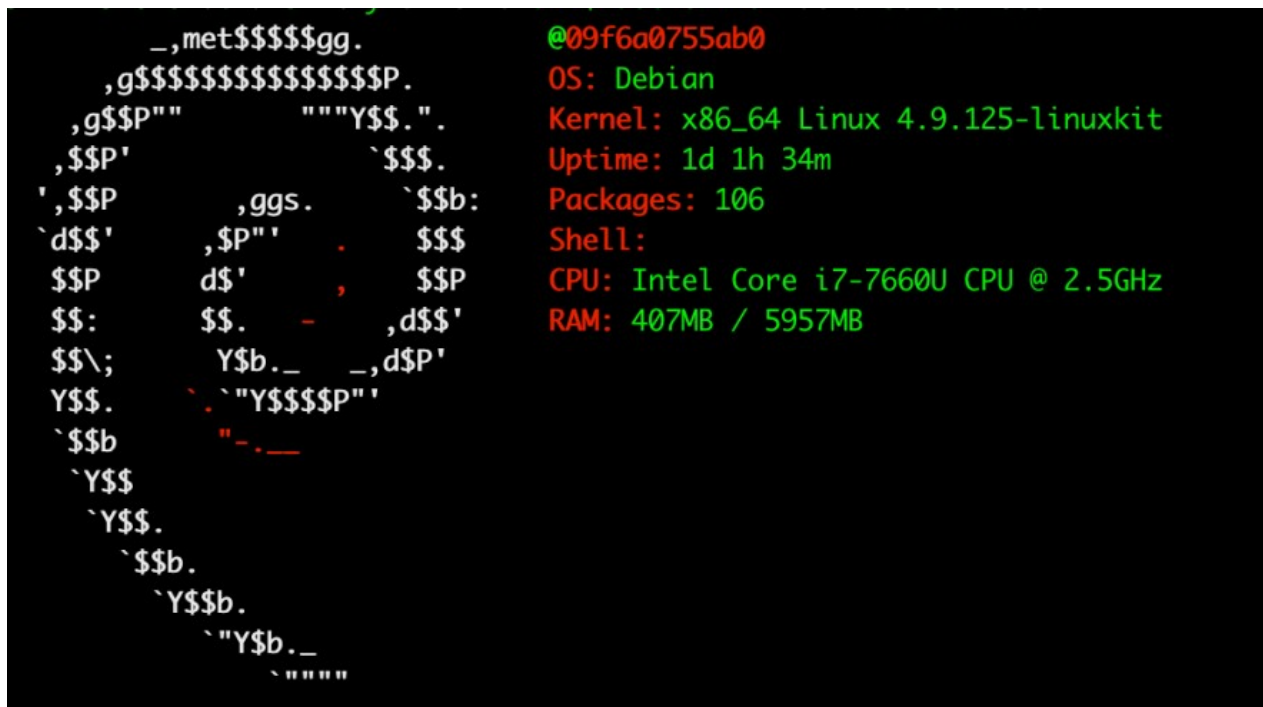
```
docker run demo
```

This will run the default executable specified in `CMD` instruction and output following.



However, you can override this executable, when running the container as

```
docker run demo screenfetch -E which will output
```



## Demo 2 - Use of ENTRYPOINT

If you want to restrict users from passing any other executable, you can specify 'ENTRYPOINT' instruction as :

```
FROM debian:jessie-slim

RUN apt-get update                                && \
    apt-get install -y --no-install-recommends    \
        cowsay                                     \
        screenfetch                                && \
    rm -rf /var/lib/apt/lists/*

ENV PATH "$PATH:/usr/games"

ENTRYPOINT ["cowsay", "Yo, Entrypoint!!"]
```

Build the image again and run this container as:

```
docker run demo
```

This will run the executable specified in `ENTRYPOINT` and output following.

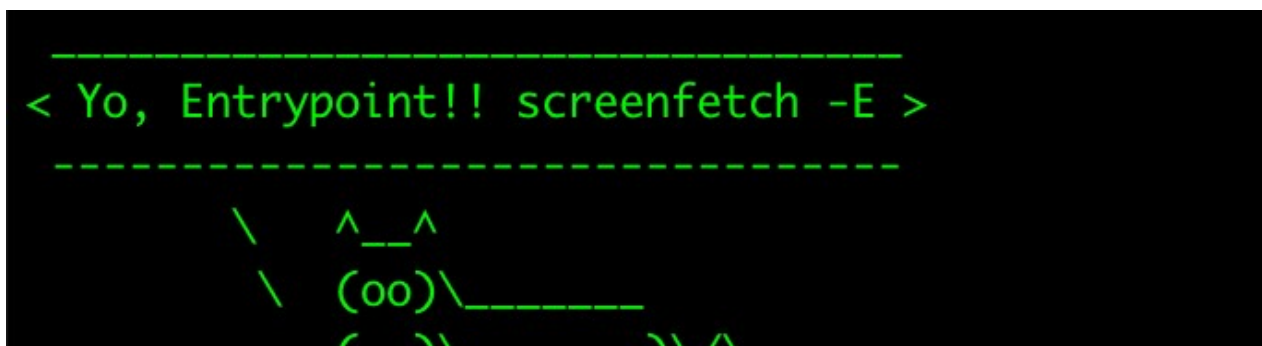


```
-----  
< Yo, Entrypoint!! >  
-----  
      ^__^  
      (oo)\_____  
      (_____)\\        )\\/  
              ||----w |  
              ||     ||
```

With `entryPoint` defined in the docker file, if you pass an executable in the docker run command, it will not take that as an executable, but, as an argument for the executable defined in the `entryPoint` instruction. So if you run following command,

```
docker run demo screenfetch -E
```

It will output:



```
-----  
< Yo, Entrypoint!! screenfetch -E >  
-----  
      ^__^  
      (oo)\_____  
      (_____)\\        )\\/  
              ||----w |  
              ||     ||
```



As expected, the arguments passed in the docker run command, is appended in the executable and argument, set in the entryPoint instruction.

### Demo 3 - Use of ENTRYPOINT and CMD together

If Entrypoint is defined, anything defined in CMD will be taken as arguments for the executable defined in Entrypoint.

```
FROM debian:jessie-slim

RUN apt-get update                                && \
    apt-get install -y --no-install-recommends    \
        cowsay                                    \
        screenfetch                               && \
    rm -rf /var/lib/apt/lists/*

ENV PATH "$PATH:/usr/games"

CMD ["Yo, CMD!!"]
ENTRYPOINT ["cowsay", "Yo, Entrypoint!!"]
```

Building the image from above docker file and running the container with `docker run demo` will output:






As you now know, the argument defined in CMD can easily be overridden by passing other arguments in docker run command, if you execute `docker run demo Overriding arg passed in cmd`, It will output:



Thus, the main point to remember is CMD is used to provide default executable and arguments, which can be overridden, while ENTRYPOINT is used to specify specific executable and arguments, to constraint the usage of image.



**Navee** + FOLLOW

Software engineer based in London. Mostly involved in #Java #Microservices #Backend  
@lasatadevi  lasatadevi




Add to the discussion




PREVIEW

SUBMIT



lagerenas 

Nov 11 

This is a great summary with clear examples. Thanks



2

REPLY

[code of conduct](#) - [report abuse](#)

Classic DEV Post from Jun 17

## Should a button communicate the current state, the intended behavior, or both?



Ben Halpern



146



60

Another Post You Might Like

# How To Incorporate Containers Into Your Daily Duties



Henry Quinn



89



2

---

Another Post You Might Like

## 5 part Docker series, beginner to master



Chris Noring

This is to tell you that I've written a 5 part Docker series that takes you all the way from beginner to proficient in Docker



1349



23



### 10 Things You Should Expect From a Container Registry

Prasanna Raghavendra - Nov 24



### GitHub PR Review, stop wasting time on code review!

BaBeuloula - Nov 24



### Ambiente Web PHP Docker, utilizando container Ubuntu

Erick Brito - Nov 23



### How to use docker multi-stage build to create optimal images for dev and production

Geshan Manandhar - Nov 24

---

[Home](#) [About](#) [Privacy Policy](#) [Terms of Use](#) [Contact](#) [Code of Conduct](#)

DEV Community copyright 2016 - 2019 