Understanding

CMD and ENTRYPOINT

with Practical Demonstrations

cmd/entrypoint commands are commonly used if we want to start a service or execute some scripts when the container starts.

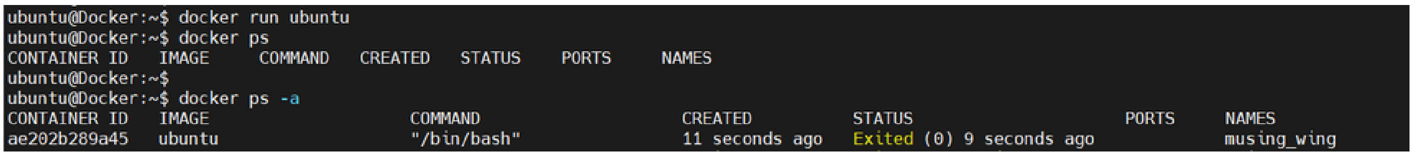
Created by Athira KK

Introduction

Docker provides a powerful way to create, deploy, and run applications using containers. In this guide, we'll explore how commands and entry points work in Docker, using a simple scenario involving an Ubuntu image.

When you run a Docker container from an Ubuntu image using the following command:

- docker run ubuntu



```
ubuntu@Docker:~$ docker run ubuntu
ubuntu@Docker:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@Docker:~$
ubuntu@Docker:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND                      CREATED          STATUS                       PORTS     NAMES
ae202b289a45   ubuntu                   "/bin/bash"                  11 seconds ago   Exited (0) 9 seconds ago               musing_wing
```

The container starts an instance of the Ubuntu image and exits immediately. If you list the running containers, you won't see it running.

- docker ps

However, if you list all containers, including stopped ones, you will see that the container is in an exited state.

- docker ps -a

(*refer the above screenshot)*

Why Does the Container Exit Immediately?

Unlike virtual machines, Docker containers are not designed to host an operating system indefinitely. Instead, they are meant to run specific tasks or processes. Examples include:

- Hosting a web server instance.
- Running an application server.
- Operating a database.
- Performing computation or analysis tasks.

A container's lifespan is tied to the lifespan of the process running inside it. If the process completes or crashes, the container exits. For instance, if a web service inside a container stops or crashes, the container will also stop.

- Containers are designed to run specific tasks or processes, not to host an operating system indefinitely.
- Container lifespan is tied to the lifespan of the process inside it. Once the process completes or crashes, the container exits.

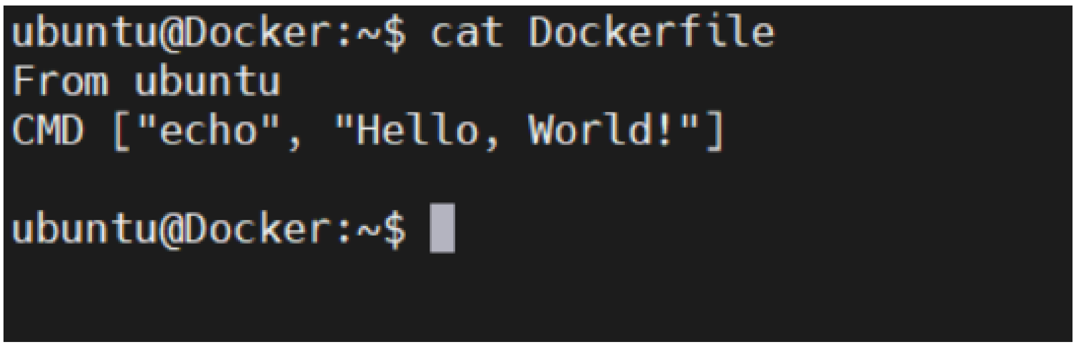
CMD Instruction

The CMD instruction defines the default command to be executed when a container starts. It can be specified in two formats: JSON format and Shell format.

Format:

- JSON format: **CMD ["executable", "param1", "param2"]** - the first element must be an executable, and the subsequent elements are its parameters.
- Shell format: **CMD executable param1 param2** - This format runs the command in the default shell /bin/sh -c.

Creating the Dockerfile:



```
ubuntu@Docker:~$ cat Dockerfile
From ubuntu
CMD ["echo", "Hello, World!"]

ubuntu@Docker:~$
```

This Dockerfile is set up to use the official Ubuntu base image and specifies a CMD instruction that echoes a simple message.

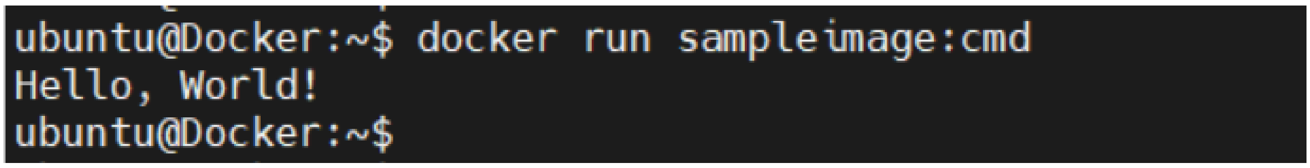
Building the Docker Image: Build the Docker image using the following command:

- docker build -t sampleimage:cmd .

The build process reads the Dockerfile, fetches the base image, and builds a new image tagged as sampleimage:cmd.

Run the Docker container using the command:

- docker run sampleimage:cmd



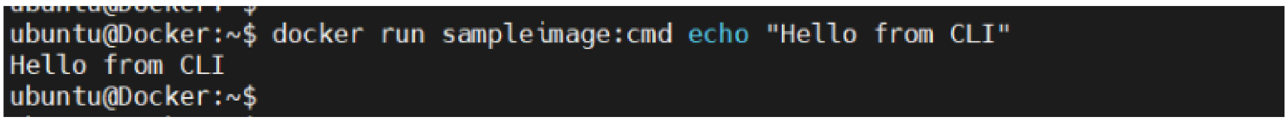
```
ubuntu@Docker:~$ docker run sampleimage:cmd
Hello, World!
ubuntu@Docker:~$
```

The container runs the default command specified in the Dockerfile, which is "Hello, World!".

****

When running a Docker container, you can override the default CMD instruction specified in the Dockerfile by appending commands to the docker run command.

- ubuntu@Docker:~$ docker run sampleimage:cmd echo "Hello from CLI" Hello from CLI echo "Hello from CLI": Overrides the default CMD instruction specified in the Dockerfile by providing a new command (echo "Hello from CLI").



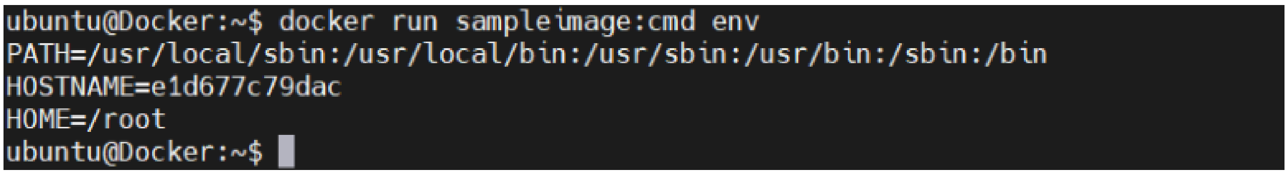
```
ubuntu@Docker:~$ docker run sampleimage:cmd echo "Hello from CLI"
Hello from CLI
ubuntu@Docker:~$
```

- By appending commands to the docker run command, you can dynamically override the default behavior defined in the Dockerfile.
- This flexibility allows for customizing container behavior based on specific requirements at runtime.

Environment Variables in Docker

Docker allows for managing and customizing environment variables both at build time (in the Dockerfile) and at runtime (via the docker run command), providing flexibility and control over container environments.

When running a Docker container, you can inspect the environment variables within the container by using the env command as demonstrated below:



```
ubuntu@Docker:~$ docker run sampleimage:cmd env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=e1d677c79dac
HOME=/root
ubuntu@Docker:~$
```

In this example:

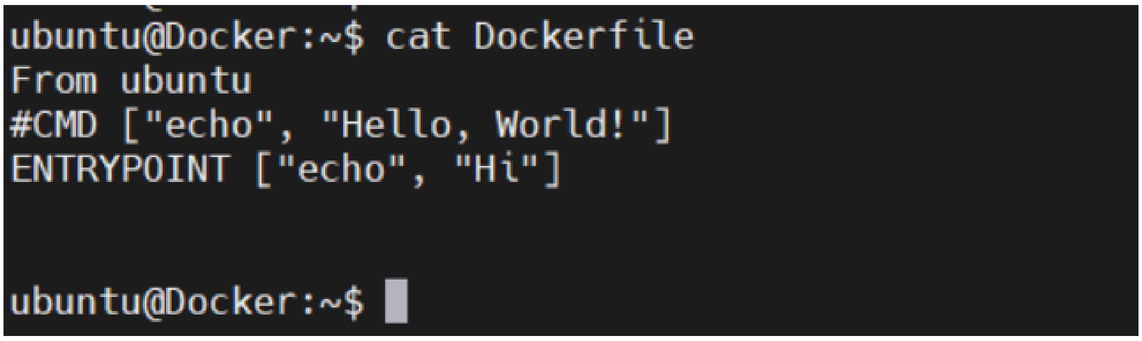
- docker run sampleimage:cmd: Starts a container based on the sampleimage image with the cmd tag.
- env: Prints out the environment variables within the container.

The env command displays environment variables such as PATH, HOSTNAME, and HOME set within the container.

Entrypoint Instruction

The ENTRYPOINT instruction in Docker specifies the default executable command for a container. It allows defining a default application or script to run when the container starts.

Creating Dockerfile with ENTRYPOINT



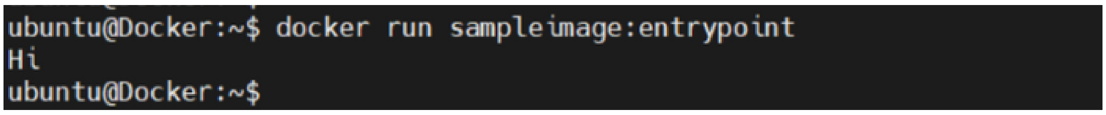
```
ubuntu@Docker:~$ cat Dockerfile
From ubuntu
#CMD ["echo", "Hello, World!"]
ENTRYPOINT ["echo", "Hi"]


ubuntu@Docker:~$
```

The Dockerfile initially specifies an ENTRYPOINT instruction with the command ["echo", "Hi"]. CMD instruction is commented-out.

Building the Docker Image with below command:
- docker build -t sampleimage:entrypoint .

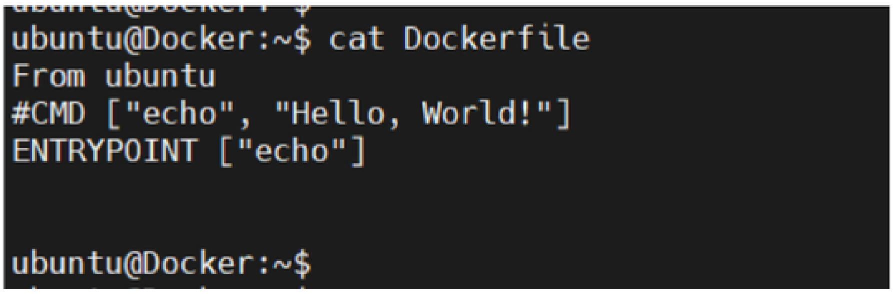
Running the Docker Container:
- docker run sampleimage:entrypoint



```
ubuntu@Docker:~$ docker run sampleimage:entrypoint
Hi
ubuntu@Docker:~$
```

- After the image is built, a container is instantiated from it using docker run sampleimage:entrypoint.
- Since no additional arguments are provided when running the container, it executes the default command specified in the ENTRYPOINT instruction, which is echo Hi.

****



```
ubuntu@Docker:~$ cat Dockerfile
From ubuntu
#CMD ["echo", "Hello, World!"]
ENTRYPOINT ["echo"]


ubuntu@Docker:~$
```

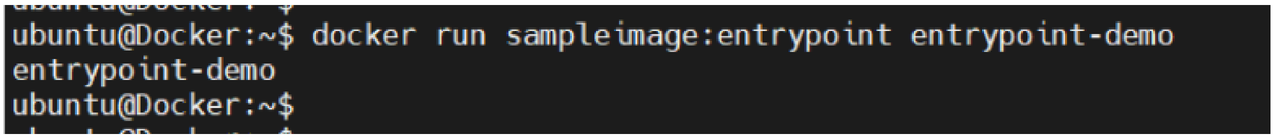
The ENTRYPOINT instruction is set to ["echo"], specifying that the default command to be executed when the container starts is echo.

Build the Docker image:

- docker build -t sampleimage:entrypoint .

Running the Docker Container:

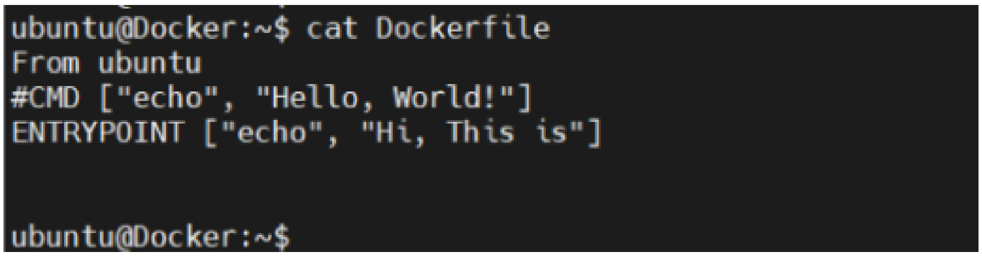
- docker run sampleimage:entrypoint entrypoint-demo



```
ubuntu@Docker:~$ docker run sampleimage:entrypoint entrypoint-demo
entrypoint-demo
ubuntu@Docker:~$
```

- Since the ENTRYPOINT instruction specifies only ["echo"], without any additional arguments, the container executes echo as its default behavior.
- The argument provided (entrypoint-demo) during container execution is not appended to any command because there's no CMD instruction to interact with.
- As a result, the output of the container is simply "entrypoint-demo", reflecting the provided argument.

****



```
ubuntu@Docker:~$ cat Dockerfile
From ubuntu
#CMD ["echo", "Hello, World!"]
ENTRYPOINT ["echo", "Hi, This is"]


ubuntu@Docker:~$
```

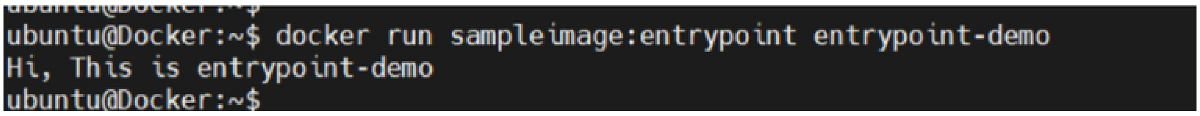
The ENTRYPOINT instruction is set to ["echo", "Hi, This is"], indicating that when the container starts, it will execute the echo command with the provided arguments.

Build the Docker image:

- docker build -t sampleimage:entrypoint .

Running the Docker Container:

- docker run sampleimage:entrypoint entrypoint-demo



```
ubuntu@Docker:~$ docker run sampleimage:entrypoint entrypoint-demo
Hi, This is entrypoint-demo
ubuntu@Docker:~$
```

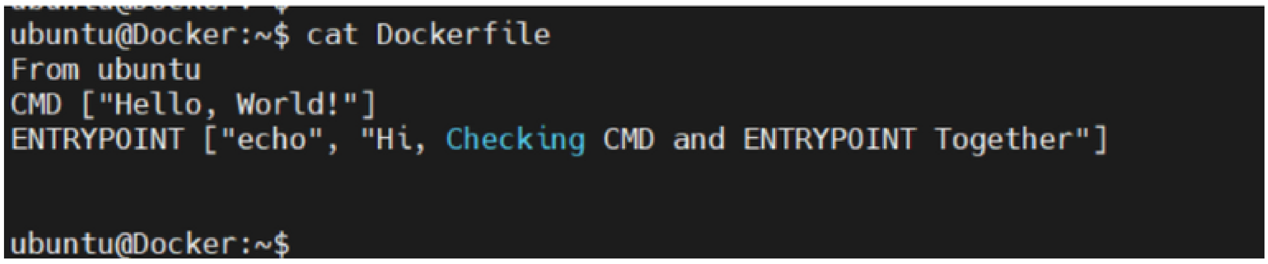
- Since there's no CMD instruction, the ENTRYPOINT instruction is executed by default. The echo command in the ENTRYPOINT is executed, and "Hi, This is" is printed.
- The provided argument entrypoint-demo is appended to the command executed by ENTRYPOINT, resulting in "Hi, This is entrypoint-demo" as the output.

ENTRYPOINT instruction defines a default command to be executed when the container starts, and any additional arguments provided during container execution will be appended to this default command.

****

Specifying CMD and ENTRYPOINT Together

When both CMD and ENTRYPOINT are specified in the Dockerfile, the ENTRYPOINT command is executed with its arguments, and then the command specified by CMD is appended to it and executed.



```
ubuntu@Docker:~$ cat Dockerfile
From ubuntu
CMD ["Hello, World!"]
ENTRYPOINT ["echo", "Hi, Checking CMD and ENTRYPOINT Together"]


ubuntu@Docker:~$
```

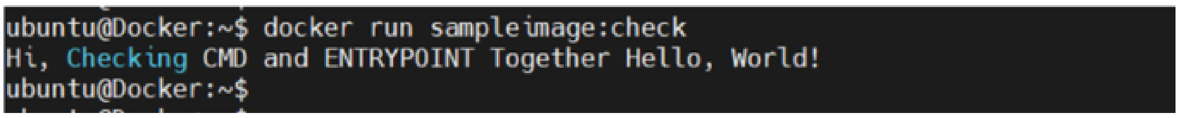
- The CMD instruction: CMD ["Hello, World!"], which sets the default command to execute when the container starts.
- The ENTRYPOINT instruction is also set: ENTRYPOINT ["echo", "Hi, Checking CMD and ENTRYPOINT Together"]. This defines the command that will always be executed when the container starts.

Build the Docker image using following command:

- docker build -t sampleimage:check .

Run the container using following command:

- docker run sampleimage:check



```
ubuntu@Docker:~$ docker run sampleimage:check
Hi, Checking CMD and ENTRYPOINT Together Hello, World!
ubuntu@Docker:~$
```

- The ENTRYPOINT command is executed with its arguments, resulting in "Hi, Checking CMD and ENTRYPOINT Together" being printed.
- Additionally, the default command specified by CMD, which is "Hello, World!", is also executed, resulting in "Hello, World!" being printed after the ENTRYPOINT output.

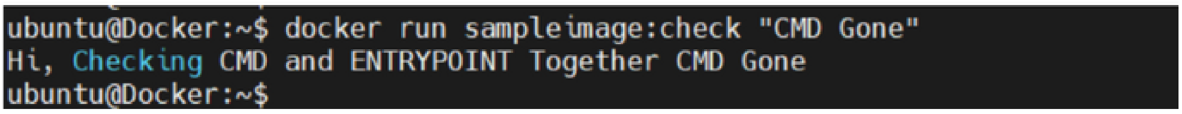
****

Docker concatenates the ENTRYPOINT command with additional arguments provided at runtime, and the CMD instruction is effectively overridden by any command-line arguments passed during container execution. This illustrates how Docker's CMD and ENTRYPOINT instructions work together and how command-line arguments can influence container behavior.

The Dockerfile used remained unchanged in both scenarios.

Run the container using following command:

- docker run sampleimage:check "CMD Gone"



```
ubuntu@Docker:~$ docker run sampleimage:check "CMD Gone"
Hi, Checking CMD and ENTRYPOINT Together CMD Gone
ubuntu@Docker:~$
```

- When you ran docker run sampleimage:check "CMD Gone", Docker combined the ENTRYPOINT command and its arguments ("echo", "Hi, Checking CMD and ENTRYPOINT Together") with the arguments provided at runtime ("CMD Gone").
- The resulting command executed inside the container is equivalent to echo "Hi, Checking CMD and ENTRYPOINT Together" "CMD Gone".
- This concatenation of commands led to the output "Hi, Checking CMD and ENTRYPOINT Together CMD Gone".

Conclusion

- CMD sets default commands and arguments but can be overridden.
- ENTRYPOINT makes the container behave like an executable and has a higher priority over CMD.
- When both are used, ENTRYPOINT specifies the command, and CMD specifies the default arguments which can be overridden.
- Additionally, ENTRYPOINT allows appending additional arguments to the specified command, providing further flexibility when running containers.