# Lab Exercise 5- Understanding CMD, RUN, and ENTRYPOINT in Dockerfile

**Objective:**

To learn the differences between CMD, RUN, and ENTRYPOINT instructions in Dockerfiles by creating and running Docker containers with different configurations.

**Prerequisites:**

- Docker installed on your machine
- Basic understanding of Docker and Dockerfile

---

**Part 1: Overview of CMD, RUN, and ENTRYPOINT**

- **RUN:** Executes commands at build time to install software, download dependencies, or configure the environment. The result is saved in the image.
- **CMD:** Specifies the default command to be executed when a container starts. It can be overridden when running a container.
- **ENTRYPOINT:** Defines the main executable for the container, which can't be easily overridden. However, additional arguments can be passed when the container starts.

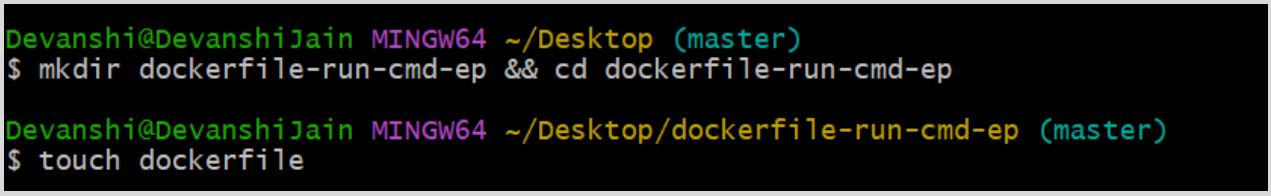
---

## Part 2: Exploring RUN Command

### 1. Create a Dockerfile with RUN:

Create a directory called dockerfile-run-cmd-entrypoint and navigate to it:

```
mkdir dockerfile-run-cmd-entrypoint && cd dockerfile-run-cmd-entrypoint
```




Create a simple Dockerfile that uses the RUN instruction:

```
# Use an official Ubuntu base image

FROM ubuntu:20.04


# Update the package repository and install curl

RUN apt-get update && apt-get install -y curl


# Print the version of curl

RUN curl --version
```

### 2. Build the Docker Image:

Build the image using the Dockerfile:

```
docker build -t run-example .
```

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker build -t run-exm .
[+] Building 65.1s (8/8) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 127B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [auth] library/ubuntu:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
 => => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
 => => sha256:13b7e930469f6d3575a320709035c6acf6f5485a76abcf03d1b92a64c09c2476 27.51MB / 27.51MB
 => => extracting sha256:13b7e930469f6d3575a320709035c6acf6f5485a76abcf03d1b92a64c09c2476
 => [2/3] RUN apt-get update && apt-get install -y curl
 => [3/3] RUN curl --version
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:bf2335af1c954a314e25c29113c39f5ac3c66661d946d09123f76c1cf746f2ac
 => => exporting config sha256:7ed7b3891c568be140acade18d1fd73e0d8f50a859f9e93cf5fa16dd362cde02
 => => exporting attestation manifest sha256:0af7a7bcc1c4d24277fc9705b4fff9f9ead90f776f8b1a6ebbfd5c6db80b57de
 => => exporting manifest list sha256:4fc7ef996a7f470c1bcb9192c901db9d2cac92d9a4c27d67d1a63a0a4cc32b39
 => => naming to docker.io/library/run-exm:latest
 => => unpacking to docker.io/library/run-exm:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ubxs3ueihq0mo9rhwzl4ob4v9
```

3. **Explanation:**

The RUN commands in this Dockerfile are executed during the image build process. The first RUN installs curl, and the second RUN command checks and prints the curl version. After the image is built, the commands executed by RUN are already baked into the image.

4. **Verify with Docker History:**

You can check the layers created by RUN using:

```
docker history run-example
```

Each RUN command creates a new layer in the image.

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker history run-exm
IMAGE          CREATED              CREATED BY                                      SIZE      COMMENT
4fc7ef996a7f   About a minute ago   RUN /bin/sh -c curl --version # buildkit        4.1kB     buildkit.dockerfile.v0
<missing>      About a minute ago   RUN /bin/sh -c apt-get update && apt-get ins…   76.6MB    buildkit.dockerfile.v0
<missing>      9 months ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      9 months ago         /bin/sh -c #(nop) ADD file:f9ee450324e6ff2c9…   81.7MB
<missing>      9 months ago         /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      9 months ago         /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      9 months ago         /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      9 months ago         /bin/sh -c #(nop)  ARG RELEASE                  0B
```

## Part 3: Exploring CMD Command

1. **Create a Dockerfile with CMD:**

Modify the Dockerfile to include the CMD instruction:

```
# Use an official Ubuntu base image
FROM ubuntu:20.04

# Install curl
RUN apt-get update && apt-get install -y curl

# Set default command to display the curl version
CMD ["curl", "--version"]
```

2. **Build the Docker Image:**

Build the Docker image again:

docker build -t cmd-example .

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker build -t cmd-exm .
[+] Building 1.8s (6/6) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 240B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
 => => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
 => CACHED [2/2] RUN apt-get update && apt-get install -y curl
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:2878460dcce369647e166edd5a1e0a87d6a852f25851abf37c61ca658be1da9c
 => => exporting config sha256:4c4882eb07a4a9727dc6545fde02922794fb072eb466c9052966e5ff7402616e
 => => exporting attestation manifest sha256:47157efd1a8a5a1c74ee4560a31dc6ab7b5510f6d4675a3f74b5d6920eb54089
 => => exporting manifest list sha256:b4547f5fce28145428df3d495553252c2498e190bc4a39ae0ed6c068a03df66e
 => => naming to docker.io/library/cmd-exm:latest
 => => unpacking to docker.io/library/cmd-exm:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/nh6da95kgkvhkh2l2n871nb86
```

3. **Run the Container:**

Run the container and see the output:

docker run cmd-example

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker run cmd-exm
curl 7.68.0 (x86_64-pc-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/z
.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets
```

The output will display the curl version as the default command defined by CMD is executed when the container starts.

4. **Override CMD:**

You can override the CMD by specifying a different command when you run the container:

docker run cmd-example echo "Hello from CMD!"

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker run cmd-exm echo "hello heloo"
hello heloo
```

This will print Hello from CMD!, showing that the CMD can be overridden at runtime.

---

**Part 4: Exploring ENTRYPOINT Command**

1. **Create a Dockerfile with ENTRYPOINT:**

Modify the Dockerfile to use ENTRYPOINT instead of CMD:

```
# Use an official Ubuntu base image
FROM ubuntu:20.04

# Install curl
RUN apt-get update && apt-get install -y curl

# Set entrypoint to curl command
ENTRYPOINT ["curl"]
```

2. **Build the Docker Image:**

Build the image with the ENTRYPOINT instruction:

```
docker build -t entrypoint-example .
```

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker build -t ep-exm .
[+] Building 3.0s (7/7) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 217B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [auth] library/ubuntu:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fa
 => => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fa
 => CACHED [2/2] RUN apt-get update && apt-get install -y curl
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:9b7735963b46c4cc67c1018f4ea138405041d89217eda772b7f33f87a7f654f9
 => => exporting config sha256:1c4cc528851433ff5b44610cd41b4c0cbf26ba5a7cdd2c04f11d053a39444528
 => => exporting attestation manifest sha256:9a3db0cfa73671a2df809e5985888842b5cd1cd45af2167fa4eb3203a02bf3
 => => exporting manifest list sha256:0ab8c8874e6710d847060647e576832a3580b00fd9d5b82cc74da16044e26805
 => => naming to docker.io/library/ep-exm:latest
 => => unpacking to docker.io/library/ep-exm:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/lg2kv6ltpb5xydue0e7hqe0tj
```

3. **Run the Container:**

When you run the container, since ENTRYPOINT is set to curl, you need to provide arguments to the curl command:

```
docker run entrypoint-example –version
```

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker run ep-exm --version
curl 7.68.0 (x86_64-pc-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/o
.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets
```

This will print the curl version because ENTRYPOINT defines the main executable (in this case, curl) and --version is passed as an argument to curl.

4. **Override ENTRYPOINT:**

Unlike CMD, the ENTRYPOINT is not easily overridden. If you try to override it using:

```
docker run entrypoint-example echo "Hello from ENTRYPOINT!"
```

It will result in an error because curl will interpret echo as an argument.

However, you can use the --entrypoint option to change the entrypoint:

```
docker run --entrypoint /bin/bash entrypoint-example -c "echo Hello from ENTRYPOINT!"
```

```
PS C:\Users\Devanshi> docker run --entrypoint /bin/bash ep-exm -c "echo Hello from ENTRYPOINT!"
Hello from ENTRYPOINT!
```

This runs the container with /bin/bash as the entrypoint, overriding the default ENTRYPOINT.

---

**Part 5: Combining CMD and ENTRYPOINT**

1. **Create a Dockerfile with Both CMD and ENTRYPOINT:**

Modify the Dockerfile to use both CMD and ENTRYPOINT:

```
# Use an official Ubuntu base image
```

```
FROM ubuntu:20.04

# Install curl
RUN apt-get update && apt-get install -y curl

# Set entrypoint to curl
ENTRYPOINT ["curl"]

# Set default arguments to --version
CMD ["--version"]
```

2. **Build the Image:**

Build the new image:

```
docker build -t combined-example .
```

3. **Run the Container:**

When you run the container without specifying any arguments, it will use the CMD as arguments to ENTRYPOINT:

```
docker run combined-example
```

The output will show the curl version, as ENTRYPOINT is curl and CMD provides --version as the argument.

4. **Override CMD Arguments:**

You can override the CMD arguments by specifying your own arguments:

```
docker run combined-example https://www.google.com
```

```
Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker build -t all-exm .
[+] Building 2.8s (7/7) FINISHED                          docker:desktop-linux
 => [internal] load build definition from dockerfile                      0.0s
 => => transferring dockerfile: 268B                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04           2.4s
 => [auth] library/ubuntu:pull token for registry-1.docker.io             0.0s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [1/2] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8f  0.0s
 => => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8f  0.0s
 => CACHED [2/2] RUN apt-get update && apt-get install -y curl            0.0s
 => exporting to image                                                    0.1s
 => => exporting layers                                                   0.0s
 => => exporting manifest sha256:0ee03da5e5727a57eafec0c50e7a4358423326f2  0.0s
 => => exporting config sha256:3931b1303fd89590cfb4c9adb9df3720f70d1fb878  0.0s
 => => exporting attestation manifest sha256:ca911e7d775cb1aca96e37fec78e  0.0s
 => => exporting manifest list sha256:b0a96b8443f5ee0af908b899703ce900c2a  0.0s
 => => naming to docker.io/library/all-exm:latest                         0.0s
 => => unpacking to docker.io/library/all-exm:latest                      0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux
/0wqiig2th20fuwgbr7h9poz4p

Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker run all-exm
curl 7.68.0 (x86_64-pc-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brot
li/1.0.7 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/zlib
nghttp2/1.40.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s
 rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile
 libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets

Devanshi@DevanshiJain MINGW64 ~/Desktop/dockerfile-run-cmd-ep (master)
$ docker run all-exm www.google.com
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0<!
doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN
"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta
 content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop
="image"><title>Google</title><script nonce="M3a6niAOdbOsUZBo2XBJwA">(function()
{var _g={kEI:'3tR1aYPdL7Kg1sQPn_yPuAg',kEXPI:'0,203004,1101199,3030760,78219,266
577,247320,53830,5231643,142,8,36811425,25381059,57131,8040,37674,2098,61250,140
67,23254,3292,34513,28334,65264,14050,7714,30734,7,2644,3050,2,25293,749,22727,2
```

This command will run curl https://www.google.com inside the container.

---

**Summary of Differences:**

- **RUN:** Executes commands during the image build process and creates layers. It is used to install packages and configure the environment.
- **CMD:** Specifies the default command to run when the container starts. It can be overridden by passing a different command when running the container.

- **ENTRYPOINT:** Specifies the main command for the container. It is harder to override but allows passing arguments from the command line. When combined with CMD, CMD provides the default arguments for ENTRYPOINT.

---

**Conclusion:**

This lab exercise demonstrates the fundamental differences between RUN, CMD, and ENTRYPOINT in Docker. Each command serves a different purpose, from image build-time configuration (RUN) to defining the container's behavior at runtime (CMD and ENTRYPOINT). Understanding these differences is crucial for building effective and flexible Docker images.