

Lab Exercise 5- Understanding CMD, RUN, and ENTRYPPOINT in Dockerfile

Objective:

To learn the differences between CMD, RUN, and ENTRYPPOINT 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 ENTRYPPOINT

- **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.
 - **ENTRYPPOINT:** 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:

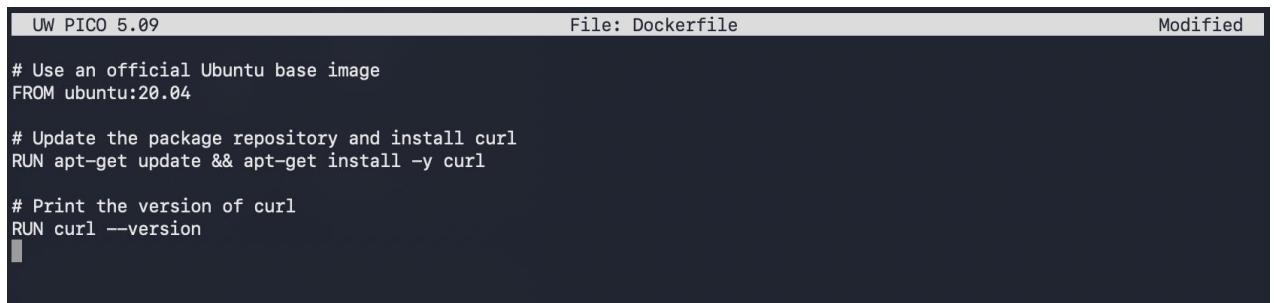
```
mkdir dockerfile-run-cmd-entrypoint && cd dockerfile-run-cmd-entrypoint
```

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
```



The screenshot shows a terminal window with the title "UW PICO 5.09". The file being edited is "Dockerfile". The code in the file is identical to the one shown in the previous code block.

```
UW PICO 5.09                               File: Dockerfile                               Modified
# 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 .
```



```
devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker build -t run-example .

[+] Building 7.6s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 239B
=> [internal] load metadata for docker.io/library/ubuntu:20.04
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> CACHED [2/3] RUN apt-get update && apt-get install -y curl
=> CACHED [3/3] RUN curl --version
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:8ca22f1175103755003827a6d9b408d0e2405e2698ada912c01b2689e3393dd0
=> => exporting config sha256:e830e7818844862d47db4a194ea143ff4db738720744d44d9501c0412ef5a0627
=> => exporting attestation manifest sha256:ce26d23256519f500eaec4b69d7c1ecbaca9b622f382d5433c549cbbf37f126a
=> => exporting manifest list sha256:8b3d9f86db55347c8fd8d9712a23ad66d3179e537d25e41fabbd3dd61571450d5
=> => naming to docker.io/library/run-example:latest
=> => unpacking to docker.io/library/run-example:latest
```

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
```

```
devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker history run-example

IMAGE      CREATED      CREATED BY
8b3d9f86db55  11 days ago  RUN /bin/sh -c curl --version # buildkit      4.1kB    buildkit.dockerfile.v0
<missing>  11 days ago  RUN /bin/sh -c apt-get update && apt-get ins...  55MB    buildkit.dockerfile.v0
<missing>  10 months ago /bin/sh -c #(nop)  CMD ["/bin/bash"]        0B
<missing>  10 months ago /bin/sh -c #(nop) ADD file:2c90d89e4dd4e1d24...  74.6MB
<missing>  10 months ago /bin/sh -c #(nop) LABEL org.opencontainers...  0B
<missing>  10 months ago /bin/sh -c #(nop) LABEL org.opencontainers...  0B
<missing>  10 months ago /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH  0B
<missing>  10 months ago /bin/sh -c #(nop) ARG RELEASE            0B
```

Each RUN command creates a new layer in the image.

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"]
```

```
File: Dockerfile

# 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 .
```

```

devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker build -t cmd-example .

[+] Building 3.8s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 232B
=> [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@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> CACHED [2/2] RUN apt-get update && apt-get install -y curl
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:4d5a6828f714c318fa17adbee8033bc7ed2c085461c1f374e23130df71e0cb9c
=> => exporting config sha256:16ac6fb34976441c532aaaf2e71f1b937c34c2cccd1d50d3b385abb7aa176b6077
=> => exporting attestation manifest sha256:4a93c615c4a480799f4b3ad7706309a6e19ad94448d5077049ecdeedfd322eaa
=> => exporting manifest list sha256:b9c8fa09fb681eb7f1af84a7bcd69e66091825486294d8803684008ce2577
=> => naming to docker.io/library/cmd-example:latest
=> => unpacking to docker.io/library/cmd-example:latest

```

3. Run the Container:

Run the container and see the output:

```
docker run cmd-example
```

```

devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run cmd-example

curl 7.68.0 (aarch64-unknown-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0 libpsl/0.2
1.0 (+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: AsyncDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-S
RP 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!"
```

```
[devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run cmd-example echo Hello from CMD!
Hello from CMD!
```

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

Part 4: Exploring ENTRYPPOINT Command

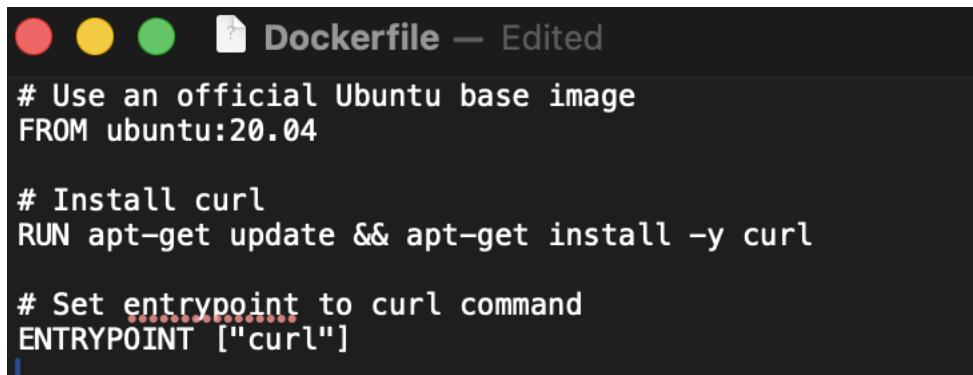
1. Create a Dockerfile with ENTRYPPOINT:

Modify the Dockerfile to use ENTRYPPOINT 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"]
```



The screenshot shows a code editor window with a dark theme. At the top, there are three circular icons (red, yellow, green) followed by the text "Dockerfile — Edited". The Dockerfile content is displayed below:

```
# 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 ENTRYPPOINT instruction:

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

```
[devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker build -t entrypoint-example .
[+] Building 3.3s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 209B
=> [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@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153
=> CACHED [2/2] RUN apt-get update && apt-get install -y curl
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:9828e2dcfd834ac1d9fa28fdfee0345a38aaa78b326176dbfce8a8461854d69f
=> => exporting config sha256:5a4c3f2f066a3fa0991f025bad1b1a1f6ee1c491a033bc049c63fae021f51b1f
=> => exporting attestation manifest sha256:3e5437a9e893e693d798ec0943a831aa657bd0e36b62e5d7f096ccc2d9a5d17b
=> => exporting manifest list sha256:cc4db3db9021c1df636a9bb2b88639151b28c4e17769459d056701fba70ee90d
=> => naming to docker.io/library/entrypoint-example:latest
=> => unpacking to docker.io/library/entrypoint-example:latest
```

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
```

```
[devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run entrypoint-example --version
curl 7.68.0 (aarch64-unknown-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0 libpsl/0.2
1.0 (+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 smt
telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-S
RP 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!"
```

```
devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run entrypoint-example echo Hello from ENTRYPOINT!
[ % Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
          Dload  Upload   Total Spent   Left  Speed
0     0    0     0    0      0  --::-- --::-- --::-- 0curl: (6) Could not resolve host: echo
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
          Dload  Upload   Total Spent   Left  Speed
0     0    0     0    0      0  --::-- --::-- --::-- 0curl: (6) Could not resolve host: Hello
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
          Dload  Upload   Total Spent   Left  Speed
[ 0     0    0     0    0      0  --::-- --::-- --::-- 0curl: (6) Could not resolve host: from
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
          Dload  Upload   Total Spent   Left  Speed
[ 0     0    0     0    0      0  --::-- --::-- --::-- 0curl: (6) Could not resolve host: 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!"
```

```
[devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run --entrypoint /bin/bash entrypoint-example
-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"]
```



```
# 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 .
```

```
devanksilswal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker build -t combined-example .  
[+] Building 2.5s (6/6) FINISHED docker:desktop-linux  
=> [internal] load build definition from Dockerfile 0.0s  
=> => transferring dockerfile: 257B 0.0s  
=> [internal] load metadata for docker.io/library/ubuntu:20.04 2.4s  
=> [internal] load .dockerignore 0.0s  
=> => transferring context: 2B 0.0s  
=> [1/2] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153 0.0s  
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2faf153 0.0s  
=> CACHED [2/2] RUN apt-get update && apt-get install -y curl 0.0s  
=> exporting to image 0.0s  
=> => exporting layers 0.0s  
=> => exporting manifest sha256:977a9097dd3bde002a04e28d09bf7e2eec649569e9d4a0c6a4b97f58f156c15c 0.0s  
=> => exporting config sha256:84295f2ad7811400536870a87c47943de1e8bfaca8523736edd236c833b5fb7e 0.0s  
=> => exporting attestation manifest sha256:e97a5efc1eaa2657daea896a9543878580fb4d2cc98012ca72c6515c438ec0e2 0.0s  
=> => exporting manifest list sha256:5ac11c22b3ed19594c5ddb9a35a341b945427fd1fec4d2a1604af87f3ea9fc99 0.0s  
=> => naming to docker.io/library/combined-example:latest 0.0s  
=> => unpacking to docker.io/library/combined-example:latest 0.0s
```

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

```
devanksilwal@devanks-MacBook-Air dockerfile-run-cmd-entrypoint % docker run combined-example

curl 7.68.0 (aarch64-unknown-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/zlib nghttp2/1.40.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftpts gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtpts 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 show the curl version, as ENTRYPPOINT 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
```

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.