

ACCA7032 Cloud Engineering Principles and Practices

Building Docker Images

Every command from the slides, step by step

This sheet lists every command in the lecture slides, in the order you run them, from an empty folder to an image on Docker Hub. Each step is tagged with the slide it comes from: **Slide 8**. A few steps are not on the slides but are needed to make the sequence work; they are tagged **Added to make it work**.

Placeholders. Replace `<dockerid>` with your Docker Hub username and `<container>` with your container's name. Type the commands without the angle brackets.

Where to type them. PowerShell or Windows Terminal on Windows, Terminal on macOS, or a WSL terminal. The commands are the same everywhere.

1. Create the project

Step 1 Make a folder and move into it

Added to make it work

```
mkdir myapp
cd myapp
```

Step 2 Create the Dockerfile

Slide 7

Save this as `Dockerfile`, with no file extension.

Dockerfile

```
FROM python:3.13-slim
WORKDIR /usr/local/app

# Install dependencies first (better cache)
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the source
COPY . .
EXPOSE 8000
CMD ["python", "app.py"]
```

Step 3 Create the two files the Dockerfile copies

Added to make it work

The Dockerfile expects `requirements.txt` and `app.py` in the same folder.

requirements.txt

```
flask==3.1.0
```

app.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello from myapp\n"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

2. Build the image

Step 4 Build and tag the image

Slide 8

The final dot is the build context: the current folder.

```
docker build -t myapp:1.0 .
```

Step 5 Check that the image exists

Slide 8

```
docker image ls
```

Look for `myapp:1.0` in the list.

3. Run and test it

Step 6 Run a container from the image

Slide 9

```
docker run --rm -p 8000:8000 myapp:1.0
```

Open `http://localhost:8000` in a browser. `--rm` removes the container when it stops, and `-p 8000:8000` maps host port to container port. This terminal now stays busy showing the app's output, so open a **second terminal** for steps 7 to 9.

Step 7 Find the container's name

Added to make it work

The slide's command gives no `--name`, so Docker chose a random one. It is in the `NAMES` column. Use it as `<container>`.

```
docker ps
```

Step 8 View the container's logs

Slide 9

```
docker logs <container>
```

Step 9 Open a shell inside the container

Slide 9

```
docker exec -it <container> sh
```

Look around with `ls` or `whoami`, then type `exit` to leave. The container keeps running.

Step 10 Stop the container

Added to make it work

In the first terminal, press **Ctrl + C**. Because of `--rm`, the container is also removed.

Shortcut: run it in the background with a name, so one terminal is enough: `docker run --rm -d --name myapp -p 8000:8000 myapp:1.0`. Then use `myapp` as `<container>`, and stop it with `docker stop myapp`.

4. Inspect the image

Step 11 Show the layers

Slides 10 and 14

Each row is one build step, with its size. Read from the bottom up.

```
docker image history myapp:1.0
```

Step 12 Show the full image metadata

Slide 10

Prints the image's settings as JSON: command, environment, ports, user, labels and digest.

```
docker image inspect myapp:1.0
```

Step 13 Inspect, naming the object type

Slide 10

```
docker inspect --type=image myapp:1.0
```

The slide says this shows “only image config fields”. In fact it prints the same full JSON as step 12: `--type=image` only tells Docker the name belongs to an image. To show only some fields, use `--format`:
`docker image inspect myapp:1.0 --format "{{.Config.Cmd}} {{.Config.ExposedPorts}}"`

5. See the build cache

Step 14 Change the code and rebuild

Slide 11

Edit the message in `app.py`, then run the build command from step 4 again.

```
docker build -t myapp:1.0 .
```

The output marks the reused steps as **CACHED**: the `pip install` step is reused, and only `COPY . .` runs again. Add `--progress=plain` to see every step in full.

6. Publish to Docker Hub

Step 15 Sign in to Docker Hub

Slide 12

```
docker login
```

Follow the sign-in in the browser, or enter your username and a personal access token.

Step 16 Tag the image with your Docker Hub name

Slide 12

Tagging adds a second name to the same image. It does not copy it.

```
docker tag myapp:1.0 <dockerid>/myapp:1.0
```

Step 17 Push the image

Slide 12

```
docker push <dockerid>/myapp:1.0
```

If the repository does not exist yet, Docker Hub creates it with your account's default privacy setting.

Step 18 Pull the image from anywhere

Slide 12

```
docker pull <dockerid>/myapp:1.0
```

Step 19 Run the pulled image to prove it works

Added to make it work

```
docker run --rm -p 8001:8000 <dockerid>/myapp:1.0
```

Open <http://localhost:8001>, then press Ctrl + C to stop it.

All slide commands at a glance

Slide	Command	What it does
8	<code>docker build -t myapp:1.0 .</code>	Build an image from this folder and tag it
8	<code>docker image ls</code>	List local images
9	<code>docker run --rm -p 8000:8000 myapp:1.0</code>	Run a container, removed when it stops
9	<code>docker logs <container></code>	Show what the container printed
9	<code>docker exec -it <container> sh</code>	Open a shell inside the container
10	<code>docker image history myapp:1.0</code>	Show the layers and their sizes
10	<code>docker image inspect myapp:1.0</code>	Show the full image metadata (JSON)
10	<code>docker inspect --type=image myapp:1.0</code>	Same output as the line above
12	<code>docker login</code>	Sign in to Docker Hub
12	<code>docker tag myapp:1.0 <dockerid>/myapp:1.0</code>	Add your Docker Hub name to the image
12	<code>docker push <dockerid>/myapp:1.0</code>	Upload the image to Docker Hub
12	<code>docker pull <dockerid>/myapp:1.0</code>	Download the image from Docker Hub
14	<code>docker image history <image:tag></code>	Show the layers of any image

Slide 7 holds the Dockerfile (step 2), and slide 6 lists its instructions: `FROM`, `WORKDIR`, `COPY/ADD`, `RUN`, `ENV`, `EXPOSE`, `USER`, `CMD/ENTRYPOINT` and `ARG/LABEL`.