An Image That Runs Anywhere

Docker: your app, and everything it needs, in one box

E190BO · AI-Amplified Prototyping

Five ideas take the idea board from your codespace into a box that runs anywhere

1. Your codespace is not the internet.
2. An image is your app and everything it needs.
3. Inside a container, `127.0.0.1` is the container.
4. What you copy in is what you ship.
5. Every run starts again from the image.

On a server, almost everything your app relied on in the codespace is gone

	In your codespace	On a server you don’t control
Python and packages	already installed	nothing, until you say what
Files	`app.db` next to `main.py`	a fresh disk, every time it starts
Who starts it	you, typing `uvicorn`	the platform, from a command you wrote down
Address and port	`localhost:8000`, through PORTS	a public address; the platform picks the port
Secrets	your `.env`, or your head	only what the platform hands it
When it breaks	the traceback in your terminal	the logs

▶ LIVE A container is your program running with its own files, name and network, on a kernel it shares

```
$ docker run --rm python:3.13-slim sh -c \
    'head -1 /etc/os-release; python --version'
PRETTY_NAME="Debian GNU/Linux 13 (trixie)"
Python 3.13.16
$ docker run --rm python:3.13-slim hostname
23cf691b5aed
$ docker run --rm python:3.13-slim uname -r
6.18.49
$ docker run --rm python:3.13-slim python -c \
    "import os; print(os.getpid())"
1
```

In the container	
files and OS	**own**
Python	**own**
name, network	**own**
processes	**own**
kernel	**shared**

The kernel is the core of the operating system: the part that runs every program.

Your codespace is a container too, built from `.devcontainer/Dockerfile` in the course template: that's why everyone's setup is identical, and the containers you run today run inside it.

A container starts in a fraction of a second, because it shares a kernel instead of booting one

	A virtual machine	A container
What it brings	a whole operating system, kernel and all	one program, and the files it needs
Starts in	tens of seconds, booting that kernel	`docker run … python -c "print(1)"`: **0.3 s**
Size	gigabytes	`python:3.13-slim`: **118MB**
Kept apart from others by	its own kernel, on pretend hardware	the shared kernel, which gives it its own view

A platform can start, stop, copy and move containers by the thousand, and yours runs the same on its machines as in your codespace.

A Dockerfile is a recipe that turns your app into an image

containers-1-dockerfile/Dockerfile

```
# Start from an image that already has Python 3.13.
FROM python:3.13-slim
# Work in /app, inside the image.
WORKDIR /app
# Copy src/, here, into /app, there.
COPY src/ .
# Run this now, at build time; keep what it installs.
RUN pip install --no-cache-dir -r requirements.txt
# Run this every time a container starts.
CMD ["uvicorn", "main:app"]
```

```
$ docker build -t idea-board .
$ docker images idea-board
REPOSITORY   TAG      SIZE
idea-board   latest   221MB
```

Build time	Run time
`FROM`, `COPY`, `RUN`	`CMD`
once, to make the image	every time a container starts
no database, no secrets	the platform's variables and database
fails in the **build log**	fails in the **deploy log**

▶ LIVE Inside a container, a server on `127.0.0.1` runs but can’t be reached

```
$ docker run --rm -p 8000:8000 idea-board
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

$ curl localhost:8000/ideas                          # from the codespace
curl: (56) Recv failure: Connection reset by peer

$ docker exec <name> python -c "import urllib.request as u; \
    print(u.urlopen('http://127.0.0.1:8000/ideas').read())"     # from inside it
b'[]'
```

Listening on `0.0.0.0` and `$PORT` makes it reachable on any port the platform picks

```
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "$PORT"]
```

```
Error: Invalid value for '--port': '$PORT' is not a valid integer.
```

containers-2-port/Dockerfile

```
FROM python:3.13-slim
WORKDIR /app
COPY src/ .
RUN pip install --no-cache-dir -r requirements.txt
ENV PORT=8000
CMD ["sh", "-c", "exec uvicorn main:app --host 0.0.0.0 --port $PORT"]
```

▶ LIVE The log always names the port the server really chose

`docker run --rm … idea-board`	The log says	`curl localhost:8000/ideas`
`-p 8000:8000`	`Uvicorn running on http://0.0.0.0:8000`	`[]`
`-e PORT=5000 -p 8000:5000`	`Uvicorn running on http://0.0.0.0:5000`	`[]`
`-e PORT=5000 -p 8000:8000`	`Uvicorn running on http://0.0.0.0:5000`	`Connection reset by peer`

`-e` sets a variable in the container, as a platform does. `-p` is *codespace port* `:` *container port*.

▶ LIVE Everything you `COPY` ships in the image, including your database and your secrets

```
$ docker run --rm idea-board ls -a /app
.
..
.env
__pycache__
app.db
main.py
requirements.txt
web
$ docker run --rm idea-board cat /app/.env
DATABASE_URL=postgresql://postgres:hunter2@db.example.com:5432/app
```

In a `.dockerignore`, a pattern needs `**/` to reach into `src/`

`.dockerignore`	What's in `/app`
(none)	`.env`, `__pycache__`, `app.db`, `main.py`, `requirements.txt`, `web`
`.env`, `*.db`	`.env`, `__pycache__`, `app.db`, `main.py`, `requirements.txt`, `web`
`**/.env`, `**/*.db`, `**/__pycache__`	`main.py`, `requirements.txt`, `web`

containers-3-ignore/.dockerignore

```
**/.env
**/*.db
**/__pycache__
```

Installing packages before copying your code lets Docker skip `pip` on every rebuild

containers-3-ignore/Dockerfile

```
FROM python:3.13-slim
WORKDIR /app
COPY src/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY src/ .
ENV PORT=8000
CMD ["sh", "-c", "exec uvicorn main:app --host 0.0.0.0 --port $PORT"]
```

Rebuilding after an edit to `main.py`	`RUN pip install …`
`COPY src/ .` first, then `pip`	`DONE 8.8s`: every package again
`COPY src/requirements.txt .`, `pip`, then `COPY src/ .`	`CACHED`

▶ LIVE A new container starts from the image, so the idea you added is gone

```
$ docker run --rm -p 8000:8000 idea-board
$ curl -X POST localhost:8000/ideas -H "Content-Type: application/json" \
    -d '{"title": "Desk fan that follows you"}'
{"id":1,"title":"Desk fan that follows you","body":"","vote_count":0}
```

Ctrl-C, and the same `docker run` again:

```
$ curl localhost:8000/ideas
[]
```

Lab 6's deploy kit is this video's `Dockerfile`, with four blanks for you to fill in

Blank in the kit's `Dockerfile`	What goes there	Where it was
`FROM ____`	an official Python 3.13 image, slim	a Dockerfile is a recipe
`COPY ____ .`	the folder your app is in	a Dockerfile is a recipe
`--host ____`	every interface	`127.0.0.1` runs but can't be reached
`--port ____`	the port the platform chose	`0.0.0.0` and `$PORT`

The kit's `.dockerignore` comes complete, with `**/`. The database stays inside the container in Lab 6, so its ideas reset on every deploy: the previous slide's idea, on purpose.