My Blood Sugar Levels

My Journey Using Docker as a Development Tool:

From Zero to Hero

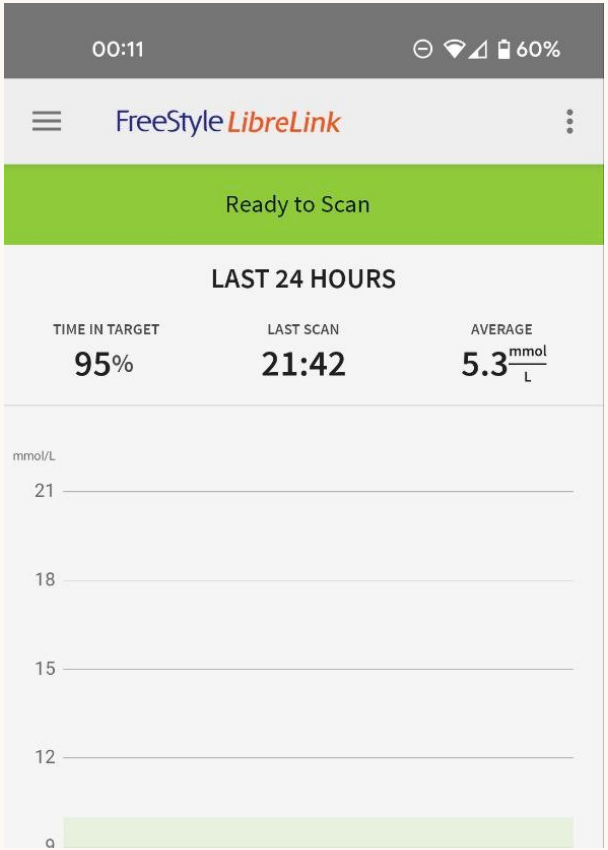
by Haseeb Majid

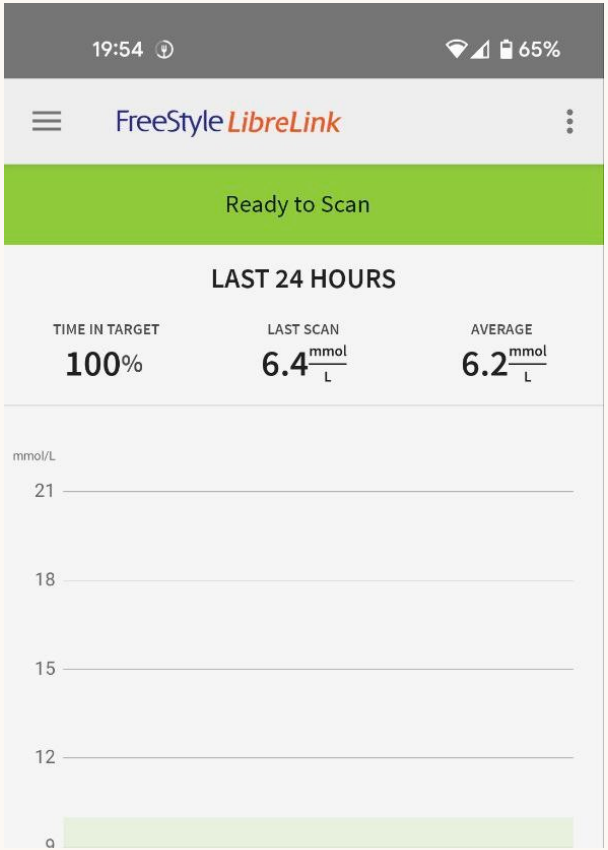
About Me

- Haseeb Majid
  - A software engineer
  - https://haseebmajid.dev
- Loves cats 🐱
- Avid cricketer 🏏 #BazBall

ZOE

- I work for ZOE 🥑
  - https://joinzoe.com
  - Personalised nutrition product
  - Health study


00:11
60%
FreeStyle LibreLink
Ready to Scan
LAST 24 HOURS
TIME IN TARGET
95%
LAST SCAN
21:42
AVERAGE
5.3 mmol/L
mmol/L
21
18
15
12
9


19:54
65%
FreeStyle LibreLink
Ready to Scan
LAST 24 HOURS
TIME IN TARGET
100%
LAST SCAN
6.4 mmol/L
AVERAGE
6.2 mmol/L
mmol/L
21
18
15
12
9

Who Is This Talk For?

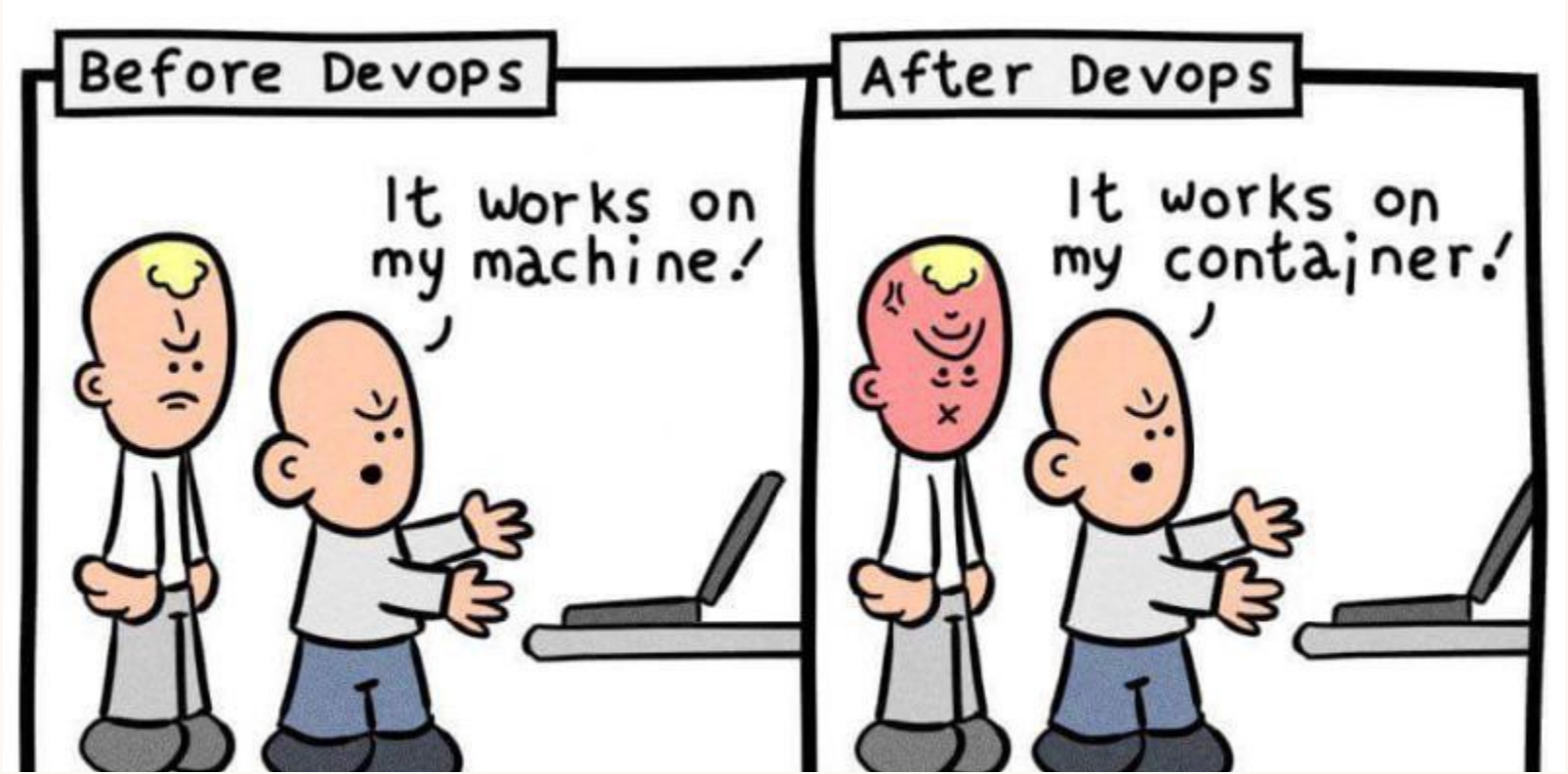
- Have used Docker
  - But not an expert
- Know basic CLI commands
- Want to use Docker in CI

Example Code

- Simple FastAPI web-service
  - Interacts with DB
- It allows us to get and add new users
- Poetry for dependency management

Why Docker?

- Reproducible builds
  - Easy setup for developers
  - OS independent


Before Devops
It works on my machine!
After Devops
It works on my container!

My First Image

```
# Dockerfile

FROM python:3.9.8

ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONPATH="/app" \
    PIP_NO_CACHE_DIR=off \
    PIP_DISABLE_PIP_VERSION_CHECK=on \
    PIP_DEFAULT_TIMEOUT=100 \
    \
    POETRY_VERSION=1.1.11 \
```

My First Image

```
4
5  ENV PYTHONUNBUFFERED=1 \
6      PYTHONDONTWRITEBYTECODE=1 \
7      PYTHONPATH="/app" \
8      PIP_NO_CACHE_DIR=off \
9      PIP_DISABLE_PIP_VERSION_CHECK=on \
10     PIP_DEFAULT_TIMEOUT=100 \
11     \
12     POETRY_VERSION=1.1.11 \
13     POETRY_HOME="/opt/poetry" \
14     POETRY_VIRTUALENVS_IN_PROJECT=true \
15     PYSETUP_PATH="/opt/pysetup" \
16     POETRY_NO_INTERACTION=1 \
```

My First Image

```
17      \
18      VENV_PATH="/opt/pysetup/.venv"
19
20  ENV PATH="$POETRY_HOME/bin:$VENV_PATH/bin:$PATH"
21
22  WORKDIR $PYSETUP_PATH
23  COPY pyproject.toml poetry.lock ./
24
25  RUN pip install poetry==$POETRY_VERSION && \
26      poetry install
27
28  WORKDIR /app
```

My First Image

```
20 ENV PATH="$POETRY_HOME/bin:$VENV_PATH/bin:$PATH"
21
22 WORKDIR $PYSETUP_PATH
23 COPY pyproject.toml poetry.lock ./
24
25 RUN pip install poetry==$POETRY_VERSION && \
26     poetry install
27
28 WORKDIR /app
29 COPY . .
30
31 CMD [ "bash", "/app/start.sh" ]
```

My First Image

```
20 ENV PATH="$POETRY_HOME/bin:$VENV_PATH/bin:$PATH"
21
22 WORKDIR $PYSETUP_PATH
23 COPY pyproject.toml poetry.lock ./
24
25 RUN pip install poetry==$POETRY_VERSION && \
26     poetry install
27
28 WORKDIR /app
29 COPY . .
30
31 CMD [ "bash", "/app/start.sh" ]
```

My First Image

```
20 ENV PATH="$POETRY_HOME/bin:$VENV_PATH/bin:$PATH"
21
22 WORKDIR $PYSETUP_PATH
23 COPY pyproject.toml poetry.lock ./
24
25 RUN pip install poetry==$POETRY_VERSION && \
26     poetry install
27
28 WORKDIR /app
29 COPY . .
30
31 CMD [ "bash", "/app/start.sh" ]
```

Let's Run It

```
docker build --tag app .
docker run --publish 80:80 app

# Access app on http://localhost
```


One container is not enough
We need to go deeper

App Dependencies

- App depends on a database
  - Dockerise it

Without Docker

```
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo systemctl start postgresql.service

sudo -u postgres createuser --interactive
sudo -u postgres createdb test
```

With Docker

```
docker run --volume "postgres_data:/var/lib/postgres
--environment "POSTGRES_DATABASE=postgres" \
--environment "POSTGRES_PASSWORD=postgres" \
--publish "5432:5432" \
  postgres:13.4
```

```
# Start Commands:
docker network create --driver bridge workspace_net
docker volume create  postgres_data
docker build -t app .
docker run --environment "POSTGRES_USER=postgres" \
  --environment "POSTGRES_HOST=postgres" \
  --environment "POSTGRES_DATABASE=postgres" \
  --environment "POSTGRES_PASSWORD=postgres" \
  --environment "POSTGRES_PORT=5432" \
  --volume "./:/app" --publish "80:8080" \
  --network workspace_network --name workspace_app
  --detach app
docker run --volume "postgres_data:/var/lib/postgre
--environment "POSTGRES_DATABASE=postgres" \
--environment "POSTGRES_PASSWORD=postgres" \
--publish "5432:5432" --network workspace_network \
--name workspace_postgres --detach postgres:13.4

# Delete Commands:
```

```
  --environment "POSTGRES_HOST=postgres" \
  --environment "POSTGRES_DATABASE=postgres" \
  --environment "POSTGRES_PASSWORD=postgres" \
  --environment "POSTGRES_PORT=5432" \
  --volume "./:/app" --publish "80:8080" \
  --network workspace_network --name workspace_app
  --detach app
docker run --volume "postgres_data:/var/lib/postgre
--environment "POSTGRES_DATABASE=postgres" \
--environment "POSTGRES_PASSWORD=postgres" \
--publish "5432:5432" --network workspace_network \
--name workspace_postgres --detach postgres:13.4

# Delete Commands:
docker stop workspace_app
docker rm workspace_app
docker stop workspace_postgres
docker rm workspace_postgres
docker network rm workspace_network
```


YOU CAN DO BETTER

Docker Compose

- Manage multiple Docker containers
- Existing tool docker-compose
  - V2 called docker compose
- Use docker compose today

```yaml
# docker-compose.yml

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    command: bash /app/start.sh --reload
    volumes:
      - ./:/app
    environment:
      - POSTGRES_USER=postgres
```

```yaml
# docker-compose.yml

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    command: bash /app/start.sh --reload
    volumes:
      - ./:/app
    environment:
      - POSTGRES_USER=postgres
```

```
 2
 3 services:
 4   app:
 5     build:
 6       context: .
 7       dockerfile: Dockerfile
 8     command: bash /app/start.sh --reload
 9     volumes:
10       - ./:/app
11     environment:
12       - POSTGRES_USER=postgres
13       - POSTGRES_HOST=postgres
14       - POSTGRES_DATABASE=postgres
```

```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    command: bash /app/start.sh --reload
    volumes:
      - ./:/app
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_HOST=postgres
      - POSTGRES_DATABASE=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_PORT=5432
```

```
    dockerfile: Dockerfile
    command: bash /app/start.sh --reload
    volumes:
      - ./:/app
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_HOST=postgres
      - POSTGRES_DATABASE=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_PORT=5432
    ports:
      - 127.0.0.1:80:80

  postgres:
```

```
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_HOST=postgres
      - POSTGRES_DATABASE=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_PORT=5432
    ports:
      - 127.0.0.1:80:80

  postgres:
    image: postgres:13.4
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
```

```
14        - POSTGRES_DATABASE=postgres
15        - POSTGRES_PASSWORD=postgres
16        - POSTGRES_PORT=5432
17      ports:
18        - 127.0.0.1:80:80
19
20    postgres:
21      image: postgres:13.4
22      volumes:
23        - postgres_data:/var/lib/postgresql/data
24      environment:
25        - POSTGRES_DATABASE=postgres
26        - POSTGRES_PASSWORD=postgres
```

```
17      ports:
18        - 127.0.0.1:80:80
19
20    postgres:
21      image: postgres:13.4
22      volumes:
23        - postgres_data:/var/lib/postgresql/data
24      environment:
25        - POSTGRES_DATABASE=postgres
26        - POSTGRES_PASSWORD=postgres
27      ports:
28        - 127.0.0.1:5432:5432
```

Run It!!!

```
docker compose up --build
```

```
docker compose down
```

Summary

- Dockerise your app
- Dockerise dependencies (DB)
- Use docker compose
  - Manage multiple containers


WHAT IF I TOLD YOU
WE CAN DO BETTER

Running Tests

- Run tests in Docker
  - pytest runner
- Consistent environment

```
docker compose run app pytest
```

```yaml
# docker-compose.yml

services:
 app:
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - postgres
    # ...

  postgres:
```

```
docker compose run app pytest
```

```
 2
 3 services:
 4  app:
 5     build:
 6       context: .
 7       dockerfile: Dockerfile
 8     depends_on:
 9       - postgres
10     # ...
11
12   postgres:
13     # ...
```

CI Pipeline

- Docker running locally
- Can we use Docker in CI?


SAY IT WORKS ON MY MACHINE

Before

```yaml
# .github/workflows/branch.yml

name: Check changes on branch

on:
  push:
    branches:
      - "*"
      - "!main"

jobs:
  test:
```

Before

```
12    test:
13      runs-on: ubuntu-latest
14      services:
15        postgres:
16          image: postgres:13.4
17          env:
18            POSTGRES_USER: postgres
19            POSTGRES_PASSWORD: postgres
20            POSTGRES_DB: postgres
21          ports:
22            - 5432:5432
23      steps:
24        - uses: actions/checkout@v3
```

Before

```
20          POSTGRES_DB: postgres
21        ports:
22          - 5432:5432
23    steps:
24      - uses: actions/checkout@v3
25      - name: Set up Python 3.9
26        uses: actions/setup-python@v3
27        with:
28          python-version: 3.9
29      - name: Install dependencies
30        run: |
31          pip install poetry=1.11.0
32          poetry install
```

Before

```
25        - name: Set up Python 3.9
26          uses: actions/setup-python@v3
27          with:
28            python-version: 3.9
29        - name: Install dependencies
30          run: |
31            pip install poetry=1.11.0
32            poetry install
33         - name: Test with pytest
34          run: |
35          export DB_USERNAME=postgres
36          export DB_PASSWORD=postgres
```

Before

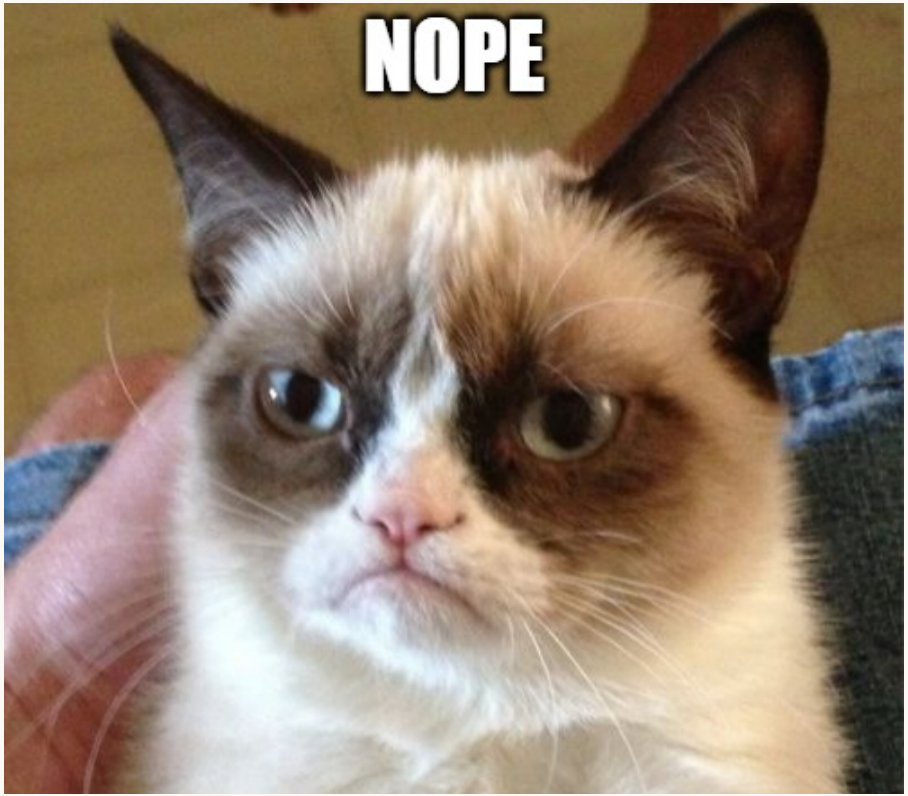
```
29        - name: Install dependencies
30          run: |
31            pip install poetry=1.11.0
32            poetry install
33         - name: Test with pytest
34          run: |
35          export DB_USERNAME=postgres
36          export DB_PASSWORD=postgres
37          export DB_HOST=postgres
38          export DB_PORT=5432
39          export DB_NAME=postgres
40          pytest
```

After

```yaml
# .github/workflows/branch.yml

name: Check changes on branch

#...

jobs:
  test:
    runs-on: ubuntu-latest
    timeout-minutes: 5
    steps:
      - uses: actions/checkout@v3
```

Summary

- Dockerise development tasks
  - Tests
  - Linting
  - DB migrations
- Use Docker on CI
  - Local environment = CI enviromnent


NOPE

Smaller Image

- Remove redundant dependencies
  - Fewer security vectors
- Less storage

```dockerfile
# Dockerfile

FROM python:3.9.8-slim

# ...

WORKDIR $PYSETUP_PATH
COPY pyproject.toml poetry.lock ./

RUN pip install poetry==$POETRY_VERSION && \
    poetry install

```

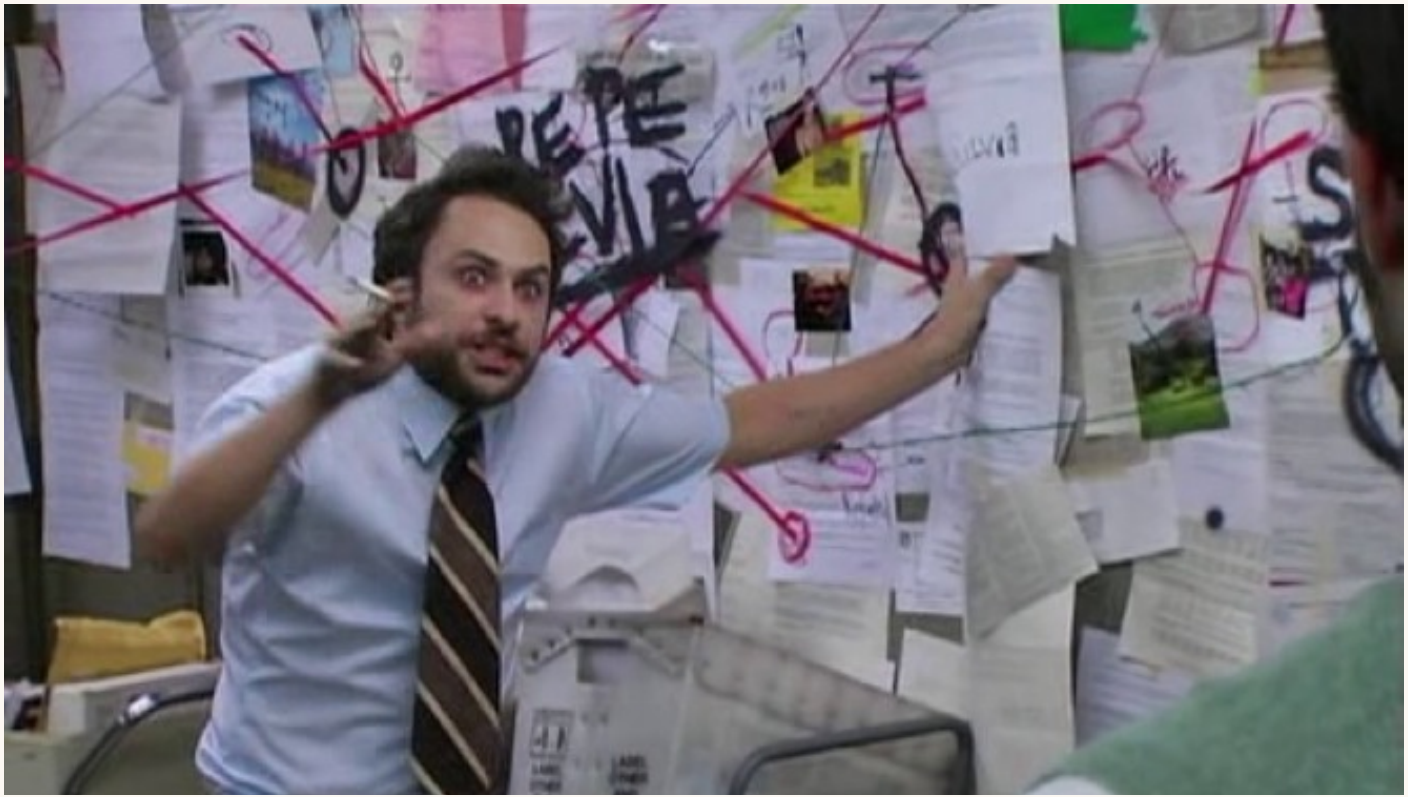
Comparison

	python:3.9.8	python:3.9.8-slim
Size	1 GB	280 MB
Build[1]	75 sec	30 sec
CI Pipeline Job	2 min 40 sec	1 min 57 sec

[1] No Cache

Summary

- Aim to use smaller base images
- Remove unnecessary dependencies
- Reduce build time



Dependencies

- Dev dependencies in Docker image
  - Don't need pytest in prod

Multistage Builds

1 Build Stage

```
FROM python:3.9.8 as builder
RUN make build-app
```

Stage Alias

Build Artefacts

```dockerfile
# Dockerfile

FROM python:3.9.8-slim as base

ARG PYSETUP_PATH
ENV PYTHONPATH="/app"
ENV PIP_NO_CACHE_DIR=off \
    PIP_DISABLE_PIP_VERSION_CHECK=on \
    PIP_DEFAULT_TIMEOUT=100 \
    \
    POETRY_VERSION=1.1.11 \
    POETRY_HOME="/opt/poetry" \
```

```
20
21
22 FROM base as builder
23
24 RUN pip install poetry==$POETRY_VERSION
25
26 WORKDIR $PYSETUP_PATH
27 COPY poetry.lock pyproject.toml ./
28
29 RUN poetry install --no-dev
30
31
```

```
30
31
32 FROM builder as development
33
34 RUN poetry install
35
36 WORKDIR /app
37 COPY . .
38
39 EXPOSE 80
40 CMD ["bash", "/app/start.sh", "--reload"]
41
42
```

```
40 CMD ["bash", "/app/start.sh", "--reload"]
41
42
43 FROM base as production
44
45 COPY --from=builder $VENV_PATH $VENV_PATH
46
47 WORKDIR /app
48 COPY . .
49
50 EXPOSE 80
51 CMD ["bash", "/app/start.sh"]
```

```
39 EXPOSE 80
40 CMD ["bash", "/app/start.sh", "--reload"]
41
42
43 FROM base as production
44
45 COPY --from=builder $VENV_PATH $VENV_PATH
46
47 WORKDIR /app
48 COPY . .
49
50 EXPOSE 80
51 CMD ["bash", "/app/start.sh"]
```

```yaml
# docker-compose.yml

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
      target: development
    command: bash /app/start.sh --reload
    depends_on:
      - postgres
    environment:
```

Comparison

	python:3.9.8-slim	Multistage[2]
Size	280 MB	200 MB
Build[1]	30 Seconds	35 seconds

[1] No Cache

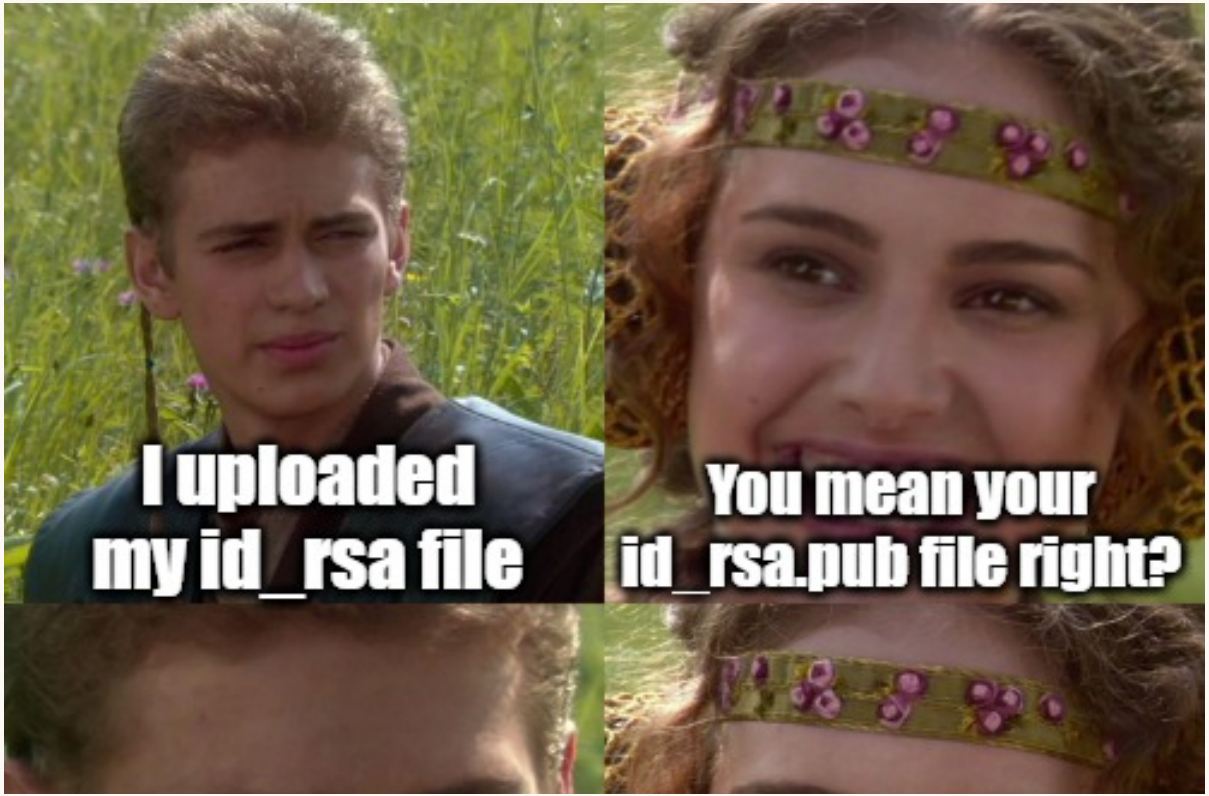
[2] Building for production target

Cache From

```
# docker-compose.yml

services:
  app:
    build:
      context: .
      target: development
      cache_from:
        - registry.gitlab.com/haseeb-slides/develop
    command: bash /app/start.sh --reload
    # ....
```

Private Deps

- Private git repository
- Inject an SSH key
  - At build time


I uploaded
my id_rsa file
You mean your
id_rsa.pub file right?

```
poetry add git+ssh@github.com:zoe/pubsub.git
```

```
1   [tool.poetry.dependencies]
2   python = "^3.9"
3   fastapi = "^0.70.0"
4   pubsub = { git = "ssh://git@github.com:zoe/pubsub.
5                rev = "0.2.5" }
6   psycopg2-binary = "^2.9.3"
7   SQLAlchemy = "^1.4.36"
8   uvicorn = "^0.17.6"
```

```
FROM base as builder

RUN apt-get update && \
    apt-get install openssh-client git -y && \
    mkdir -p -m 0600 \
    ~/.ssh && ssh-keyscan github.com >> ~/.ssh/know
    pip install poetry==$POETRY_VERSION

WORKDIR $PYSETUP_PATH
COPY poetry.lock pyproject.toml ./

RUN --mount=type=ssh poetry install --no-dev
```

```
1  FROM base as builder
2
3  RUN apt-get update && \
4      apt-get install openssh-client git -y && \
5      mkdir -p -m 0600 \
6      ~/.ssh && ssh-keyscan github.com >> ~/.ssh/know
7      pip install poetry==$POETRY_VERSION
8
9  WORKDIR $PYSETUP_PATH
10 COPY poetry.lock pyproject.toml ./
11
12 RUN --mount=type=ssh poetry install --no-dev
```

First add our ssh key

```
ssh-add ~/.ssh/id_rsa
```

Then we can do

```
docker compose build --ssh default
```

CI Changes

```yaml
# .github/workflows/branch.yml

jobs:
  # ...
  test:
    # ...
    steps:
      - uses: actions/checkout@v3
      - uses: webfactory/ssh-agent@v0.5.4
        with:
          ssh-private-key: ${{ secrets.PRIVATE_SSH_
      - name: Build Image
        run: docker compose build --ssh default
      - name: Run Tests
        run: docker compose run app pytest
```

CI Changes

```yaml
# .github/workflows/branch.yml

jobs:
  # ...
  test:
    # ...
    steps:
      - uses: actions/checkout@v3
      - uses: webfactory/ssh-agent@v0.5.4
        with:
          ssh-private-key: ${{ secrets.PRIVATE_SSH_
      - name: Build Image
        run: docker compose build --ssh default
      - name: Run Tests
        run: docker compose run app pytest
```

Comparison

	python:3.9.8-slim[2]	Multistage[3]
Size	400 MB	200 MB
Build[1]	39 Seconds	46 seconds

[1] No Cache

[2] Assuming there was no multistage build

[3] Building for production target

Summary

- Use multistage builds
  - Slimmer production images
- Leverage SSH injection
  - During build time

What Did We Do?

- Dockerised app/deps
- Used docker compose
- Used Docker for dev tasks
- Multistage builds

Even Better

- Common base image
- Makefile
- Devcontainer in VSCode
- Docker Python interpreter in Pycharm

Any Questions?

- Code: https://gitlab.com/hmajid2301/talks/docker-as-a-dev-tool
- Slides: https://docker-as-a-dev-tool.haseebmajid.dev/

Extra Reading

- Breaking Down Docker by Nawaz Siddiqui
- Announcing Compose V2 General Availability
- Caching Docker layers on serverless build hosts with multi-stage builds
- Using Alpine can make Python Docker builds 50× slower

Useful Tools

- Dive
- Anchore image scan

Appendix

- Container meme from
- Memes generated at
- Hand illustration
- People Hands Up Vectors by Vecteezy
- FlatIcon
  - Box icons created by Freepik - Flaticon
  - Arrow icons created by rang - Flaticon
  - One icons created by Hight Quality Icons - Flaticon
  - Two icons created by Hight Quality Icons - Flaticon