PYTHON IMAGES

2 STAGES BUILD

Given a Dockerfile with requirements txt file defining packages to be installed

```
FROM python:3.10-slim as compiler
ENV PYTHONUNBUFFERED 1

WORKDIR /app/

RUN python -m venv /opt/venv
# Enable venv
ENV PATH="/opt/venv/bin:$PATH"

ADD requirements.txt /app/
RUN pip install -r requirements.txt

FROM python:3.10-slim as runner
WORKDIR /app/
COPY --from=compiler /opt/venv /opt/venv

# Enable venv
ENV PATH="/opt/venv/bin:$PATH"
COPY src .

CMD ["python", "run.py"]
```

Sometimes extra libs are needed to compile code. 2 stages build help reducing the final size of image used at runtime.

https://github.com/ynov-campus-sophia/devops-B3-2023/tree/main/docker/two-stages

VIRTUAL ENV TOOLS

virtualenv, **pipenv**, **venv** all serves the same purpose, isolating your third party packages installed by Python package manager pip in a virtual environment, separated from your system level Python dependencies.

Virtualenv:

It is the by far the most used virtual environment tool to manage your pip packages in an isolated environment.

It works by installing a bunch of files in a directory (e.g. env/), and then modifying the PATH environment variable to prefix it with a custom bin directory (e.g. env/bin/).

To use it, all you have to do is install it, create an environment, then activate it, and you are ready to install the pip packages.

Pipenv

pipenv is similar to virtualenv, it has one extra feature, which is Pipfile, Pipfile is similat to what we see as packages.json in npm.

Pipfile contains all the installed packages in the current environment, and automatically updates itself if you install a new package inside the same environment.

Venv

It serves the same purpose as virtualenv, but only has a subset of its features.
In many Linux distributions, venv comes preinstalled with Python 3.

PIPENV – local dev

On local dev machine check python version (install others if needed):

```
python --version
Python 3.8
```

Install pipenv

```
pip install pipenv
```

Activate venv

```
python3.8 -m pipenv shell --python /usr/bin/python3.8
```

Install packages defined in Pipfile → a lockfile is created!

```
Pipenv install
```

PIPENV

Given a Dockerfile with requirements txt file defining packages to be installed

```
FROM python:3.8-slim as builder
LABEL maintainer="aureliano.buendias@outlook.com"

RUN pip install --no-cache-dir --upgrade pip
RUN pip install pipenv

WORKDIR /usr/local/src
ADD Pipfile.lock Pipfile /usr/local/src/
RUN PIPENV_VENV_IN_PROJECT=1 pipenv sync
COPY src .

ENTRYPOINT ["./.venv/bin/python"]
CMD ["run.py"]
```

https://github.com/ynov-campus-sophia/devops-B3-2023/tree/main/docker/pipenv

SECURE

Alpine is a lightweight distribution secure by default

```
FROM python:3.8-alpine as builder

LABEL maintainer="aureliano.buendias@outlook.com"

# Install build dependancies
RUN apk update && apk add --upgrade --no-cache g++ make

RUN pip install --no-cache-dir --upgrade pip
RUN pip install pipenv

WORKDIR /usr/local/src

ADD Pipfile.lock Pipfile /usr/local/src/

RUN PIPENV_VENV_IN_PROJECT=1 pipenv sync

FROM python:3.8-alpine as runtime

# Create the user ynov
RUN addgroup --gid 1000 ynov
RUN adduser --disabled-password --home /home/ynov --ingroup ynov \
            --uid 1000 ynov

USER ynov
WORKDIR /home/ynov
COPY src .
COPY --from=builder /usr/local/src/.venv /home/ynov/.venv

ENTRYPOINT ["./.venv/bin/python"]
CMD ["run.py"]
```

https://github.com/ynov-campus-sophia/devops-B3-2023/tree/main/docker/secure