CONTAINERIZING PYTHON

Crafting Better and Efficient Containers

AVIK BASU

ABOUT ME

- Introduced to Python in 2013
- Background in Data Science
- Love RPG games
- Hopefully a yoga instructor in the near future!
- Staff MLE at Intuit
- Leading AIOps efforts to detect incidents faster

avikbasu93@gmail.com

LinkedIn: https://www.linkedin.com/in/avik-basu
GitHub: https://github.com/ab93

MOTIVATION

- Docker images and containers are the industry standard
- Building an image for Python apps can be initially straightforward
- It can be tricky to optimize it for,
  - Faster builds
  - Smaller image size
- More so if the project has non-Python dependencies

CONTAINERIZATION BASICS

Docker Container

- Packaged application with its dependencies.
- Isolated from other processes and environments.

Docker Image

- Instruction template for running a container.
- Composed of layers that can add/remove/update files
- Written in a Dockerfile

Benefits of Containers

- Portability
- Isolation
- Scalability
- Lightweight compared to VMs

WHY SHOULD WE CARE ABOUT MAKING BETTER IMAGES?

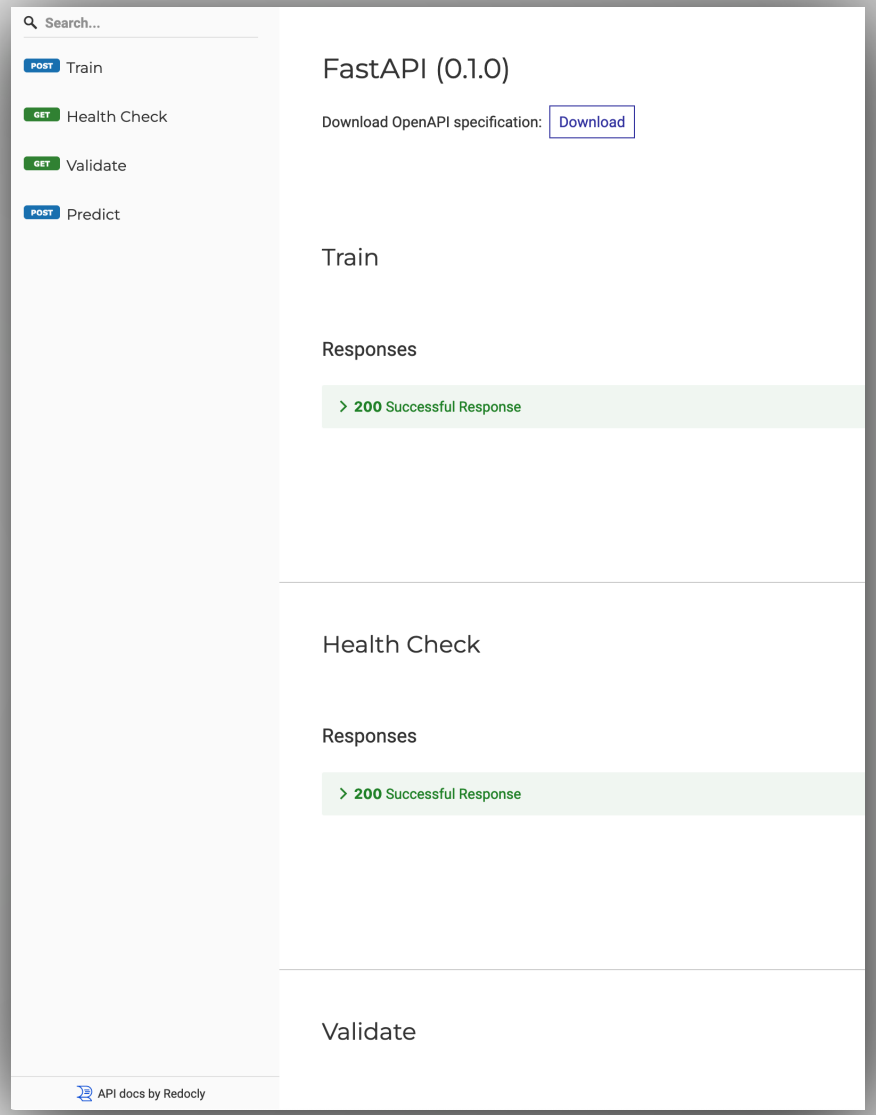
- Slower build times
  - Decreases developer productivity
  - Difficult to fail fast
- Larger image sizes
  - Requires more storage
  - Longer download times
- Security Risks
  - Older dependencies can create vulnerabilities
- Performance Overhead

HOW DO WE MEASURE EFFICIENCY?

1. Uncompressed image size
2. Initial (very first) build time
3. Rebuild time after a code change
4. Rebuild time after a dependency change

OUR EXAMPLE APPLICATION

- FastAPI based ML app using MNIST dataset
- Uses a Convolutional network written in PyTorch
- Includes a simple C++ extension



C++ EXTENSION

```cpp
#include <torch/extension.h>


int64_t argmax(torch::Tensor y) {
  return torch::argmax(y).item().toInt();

}


PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
  m.def("argmax", &argmax, "Argmax");
}
```

C++ extension file

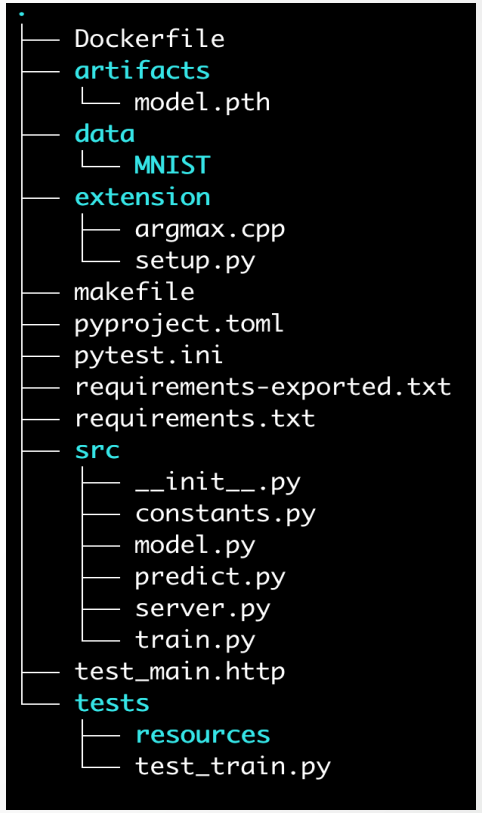
```python
def classify_image(image: Image):
    x = pil_to_tensor(image)
    tx = Compose([ToImage(), ToDtype(torch.float32)])
    x = tx(x).unsqueeze(0)
    model = torch.load(
        os.path.join(ARTIFACT_DIR, "model.pth")
    )
    with torch.no_grad():
        out = model(x)
        prediction = argmax(out)
    return prediction
```

Function usage

INITIAL DOCKERFILE

requirements.txt

fastapi==0.110.0
uvicorn==0.29
python-multipart==0.0.9
torch==2.2.1
torchvision==0.17.1
black==24.3.0
pytest==8.1.1



INITIAL DOCKERFILE

The Good

- Uses a .dockerignore file
- Based from a good image
- It works!

Image Size	Build time (no cache)	Rebuild (no change)	Rebuild time (with code change)	Rebuild time (with dependency change)
1.85 GB	55 sec	3 sec	55 sec	55 sec

```dockerfile
# Base
FROM python:3.12-bookworm as initial

ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# copy src files
COPY ./src /app/src

# copy requirements and extension dir
COPY ./requirements.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements.txt

# install non-python dependencies
RUN python setup.py install

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

INITIAL DOCKERFILE

The Bad

- Dependencies are reinstalled for every code change
- Uncompressed image size of 1.85 GB!
- Builds are non-reproducible

```
# Base
FROM python:3.12-bookworm as initial

ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# copy src files
COPY ./src /app/src

# copy requirements and extension dir
COPY ./requirements.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements.txt

# install non-python dependencies
RUN python setup.py install

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 1: ORDER MATTERS

The Good

Changes in source files do not trigger dependency installation

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	1.85 GB	55 sec	55 sec	55 sec
Now	1.85 GB	55 sec	3 sec	55 sec

```dockerfile
# Better order
FROM python:3.12-bookworm as better_order

ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# copy requirements and extension dir
COPY ./requirements.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements.txt

# install non-python dependencies
RUN python setup.py install

# copy src files
COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 1: ORDER MATTERS

The Bad

- Builds are still non-reproducible
- Uncompressed image size of 1.85 GB!

```dockerfile
# Better order
FROM python:3.12-bookworm as better_order

ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# copy requirements and extension dir
COPY ./requirements.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements.txt

# install non-python dependencies
RUN python setup.py install

# copy src files
COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 2: PIN DEPENDENCIES + DISABLE PIP CACHE

The Good

- Reproducible builds
  Using a package manager tool, e.g., poetry, hatch, etc. to pin transitive dependencies
- Separate Dev and Main dependencies
- Disable pip cache

The Bad

- Uncompressed image size of 1.71 GB!

```
# 1. Pin Transitive Dependencies
# 2. Disable pip cache
FROM python:3.12-bookworm as poetryexport

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

WORKDIR /app

COPY ./requirements-exported.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements-exported.txt

# install non-python dependencies
RUN python setup.py install

COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 2: PIN DEPENDENCIES + DISABLE PIP CACHE

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	1.85 GB	55 sec	3 sec	55 sec
Now	1.71 GB	50 sec	3 sec	49 sec

```
# 1. Pin Transitive Dependencies
# 2. Disable pip cache
FROM python:3.12-bookworm as poetryexport

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

WORKDIR /app

COPY ./requirements-exported.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements-exported.txt

# install non-python dependencies
RUN python setup.py install

COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 3: SMALLER BASE IMAGE

The Ugly

- Build error for the C++ extensions
- Slim base image does not have a C++ compiler to build the extension

```
# Use slim base image
FROM python:3.12-slim-bookworm as slimmed_err

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

WORKDIR /app

COPY ./requirements-exported.txt ./extension /app/

# install python dependencies
RUN pip install -r requirements-exported.txt

# install non-python dependencies
# this will FAIL!
RUN python setup.py install

COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 3: SMALLER BASE IMAGE

The Good

- Smaller image
  Considerable amount of size reduction
- Lower base image download time

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	1.71 GB	50 sec	3 sec	49 sec
Now	1.12 GB	70 sec	3 sec	52 sec

```dockerfile
# 1. Use slim base image
# 2. Install C++ compiler
# 3. Remove compiler after build
FROM python:3.12-slim-bookworm as slimmed

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

RUN apt-get update \
    && apt-get install -y g++

WORKDIR /app

COPY ./requirements-exported.txt ./extension /app/

RUN pip install -r requirements-exported.txt
RUN python setup.py install
RUN apt-get purge -y --auto-remove g++

COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 3: SMALLER BASE IMAGE

The Bad

Increase in the initial build time due to compiler installations

The Spooky

Are we really removing the g++ compiler in the right way?

```
# 1. Use slim base image
# 2. Install C++ compiler
# 3. Remove compiler after build
FROM python:3.12-slim-bookworm as slimmed

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

RUN apt-get update \
    && apt-get install -y g++

WORKDIR /app

COPY ./requirements-exported.txt ./extension /app/

RUN pip install -r requirements-exported.txt
RUN python setup.py install
RUN apt-get purge -y --auto-remove g++

COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 4: COMBINING LAYERS

The Good

- Compiler package removal happening in the right way
- Smaller image
  Considerable amount of size reduction

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	1.12 GB	70 sec	3 sec	52 sec
Now	870 MB	65 sec	3 sec	63 sec

```
# Combine layers
FROM python:3.12-slim-bookworm as single_layer

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

WORKDIR /app

# copy requirements and extension dir
COPY ./requirements-exported.txt ./extension /app/

# build dependencies
RUN apt-get update \
    && apt-get install -y g++ \
    && pip install -r requirements-exported.txt \
    && python setup.py install \
    && apt-get purge -y --auto-remove g++

# copy src files
COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 4: COMBINING LAYERS

The Not so Good

Any change in dependencies will trigger all 5 of those RUN commands

```
# Combine layers
FROM python:3.12-slim-bookworm as single_layer

# disable pip cache
ENV PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_NO_CACHE_DIR=1

WORKDIR /app

# copy requirements and extension dir
COPY ./requirements-exported.txt ./extension /app/

# build dependencies
RUN apt-get update \
    && apt-get install -y g++ \
    && pip install -r requirements-exported.txt \
    && python setup.py install \
    && apt-get purge -y --auto-remove g++

# copy src files
COPY ./src /app/src

EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 5: MULTISTAGE BUILD

The Good

- Dependencies are installed in a virtualenv
- The **builder** image takes care of installing dependencies
- The **runner** image only has the required Python environment to run the app
- Builder can be the non-slim image
- Minor improvement in size

```
FROM python:3.12-bookworm as builder

ENV VIRTUAL_ENV=/opt/venv \
    PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

RUN python -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

COPY requirements-exported.txt extension ./

RUN pip install setuptools \
    && pip install -r requirements-exported.txt \
    && python setup.py install
```

```
FROM python:3.12-slim-bookworm AS multistage_runner_1

ENV VIRTUAL_ENV=/opt/venv
COPY --from=builder ${VIRTUAL_ENV} ${VIRTUAL_ENV}
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

WORKDIR /app
COPY ./src /app/src


EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 5: MULTISTAGE BUILD

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	870 MB	65 sec	3 sec	63 sec
Now	832 MB	54 sec	4 sec	51 sec

Nitpick Maybe?

- Packages are downloaded every time that layer gets triggered

```
FROM python:3.12-bookworm as builder

ENV VIRTUAL_ENV=/opt/venv \
    PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

RUN python -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

COPY requirements-exported.txt extension ./

RUN pip install setuptools \
    && pip install -r requirements-exported.txt \
    && python setup.py install

---

FROM python:3.12-slim-bookworm AS multistage_runner_1

ENV VIRTUAL_ENV=/opt/venv
COPY --from=builder ${VIRTUAL_ENV} ${VIRTUAL_ENV}
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

WORKDIR /app
COPY ./src /app/src


EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

OPTIMIZATION 6: CACHE MOUNT

The Good

- The Pip cache is mounted onto the RUN cache *avoids dependency download from internet*

	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
Before	832 MB	54 sec	4 sec	51 sec
Now	832 MB	56 sec	1 sec	38 sec

```
FROM python:3.12-bookworm as mntcache_builder

ENV VIRTUAL_ENV=/opt/venv \
    PIP_DEFAULT_TIMEOUT=100 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

RUN python -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

COPY requirements-exported.txt extension ./

RUN --mount=type=cache,target=/root/.cache/pip pip install setuptools \
    && pip install -r requirements-exported.txt \
    && python setup.py install
```

```
FROM python:3.12-slim-bookworm AS multistage_runner

ENV VIRTUAL_ENV=/opt/venv
COPY --from=mntcache_builder ${VIRTUAL_ENV} ${VIRTUAL_ENV}
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

WORKDIR /app
COPY ./src /app/src


EXPOSE 80
CMD ["uvicorn", "src.server:app", "--host", "0.0.0.0", "--port", "80"]
```

TO RECAP

opt no.	Dockerfile	Image Size	Build time (no cache)	Rebuild time (with code change)	Rebuild time (with dependency change)
0	Initial	1.85 GB	55 sec	55 sec	55 sec
1	Layer Ordering	1.85 GB	55 sec	3 sec	55 sec
2	Pin deps & Disable pip cache	1.71 GB	50 sec	3 sec	49 sec
3	Smaller Base Image	1.12 GB	70 sec	3 sec	52 sec
4	Combining Layers	870 MB	65 sec	3 sec	63 sec
5	Multi-stage Build	832 MB	54 sec	4 sec	51 sec
6	Multi-stage Build with Cache mount	832 MB	56 sec	1 sec	38 sec

OTHER BEST PRACTICES

- Always use a Python-specific *.dockerignore* file
- Separate Dev and Prod dependencies
- Use the latest Debian/Ubuntu/Redhat distribution to base from
- Try to avoid specifying the Python patch version in the base image, e.g., *3.12-bookworm* instead of *3.12.2-bookworm*
- CPU specific vs GPU specific image

THANK YOU! 🙏🏽

Code: https://github.com/ab93/dockerdemo