Building Python Applications with Docker

Moshe Zadka

PyTexas 2017

Introduction

- Clarify terms

Introduction

- Clarify terms
- Common mistakes

Introduction

- Clarify terms
- Common mistakes
- Best practices

What are containers?

Share-less processes

What is Docker?

- client

What is Docker?

- client
- server (containerd)

What is Docker?

- client
- server (containerd)
- runner (runc)

What is Docker?

- client
- server (containerd)
- runner (runc)
- hub

What is a Container image?

- Tarball...

What is a Container image?

- Tarball...
- that looks like a linux root

What is a Container image?

- Tarball...
- that looks like a linux root
- Will also specify some metadata (e.g., entrypoint)

What is a Dockerfile?

Builds a Docker image

```
FROM ....
COPY from-context
RUN some command
...
```

Python application

catx/app.py

```
def hello_world(request):
    return response.Response('California␣<3␣Texas')

with config.Configurator() as cfg:
    cfg.add_route('hello', '/')
    cfg.add_view(hello_world, route_name='hello')
    app = cfg.make_wsgi_app()
```

catx/__main__.py

```
simple_server.make_server('0.0.0.0', 6543, app.app)
```

What's bad?

bad.docker

```
FROM debian:latest
RUN apt-get update
RUN apt-get -y install python3 python3-pip
RUN pip3 install pyramid
COPY catx /app/catx
ENV PYTHONPATH /app/
CMD python3 -m catx
```

What's wrong?

- latest

What's wrong?

- latest
- Layer explosion

What's wrong?

- latest
- Layer explosion
- Random system packages

What's wrong?

- latest
- Layer explosion
- Random system packages
- Random Python packages

What's wrong?

- latest
- Layer explosion
- Random system packages
- Random Python packages
- PYTHONPATH

What's wrong?

- latest
- Layer explosion
- Random system packages
- Random Python packages
- PYTHONPATH
- Shipping build environment

What's wrong?

- latest
- Layer explosion
- Random system packages
- Random Python packages
- PYTHONPATH
- Shipping build environment
- Using reference WSGI server

Base images

Choose foundation wisely (and stabley)

Fork tagging

Renaming images for posterity

Fork tagging

```
IMAGE = 'pypy:3-slim'
TAG = (datetime.datetime.now()
                .isoformat().replace(':', '-')
                            .replace('.', '-'))
NAME = 'moshez/pypy3:' + TAG
cmd('docker', 'pull', IMAGE)
cmd('docker', 'tag', IMAGE, NAME)
cmd('docker', 'push', NAME)
```

Python Artifacts

sdist

- What `python setup.py sdist` creates
- Basically just a tarball of sources

Python Artifacts

wheel

- Fully built and ready
- Reliable installation

Python Artifacts

pex

- Python executable
- Installation is ”copy”
- Does not work well with PyPy

Python Artifacts

Virtual environments

- Location specific
- Can be used with `dh_virtualenv`

What is Multistage Build?

- Build images sequentially

What is Multistage Build?

- Build images sequentially
- Throw away all except last

What is Multistage Build?

- Build images sequentially
- Throw away all except last
- Copy previous image

Multistage Build

```
FROM source as name1
...
FROM other-source
...
COPY --from=name1 ...
...
ENTRYPOINT [...]
```

Better Version 1

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199 \
     as builder
```

Better Version 1

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199 \
     as builder

RUN pypy3 -m venv /appenv
RUN /appenv/bin/pip install twisted pyramid
```

Better Version 1

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199 \
     as builder

RUN pypy3 -m venv /appenv
RUN /appenv/bin/pip install twisted pyramid

COPY catx/ /mnt/src/catx/
COPY setup.py /mnt/src/
RUN /appenv/bin/pip install /mnt/src/
```

Better Version 1

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199 \
     as builder

RUN pypy3 -m venv /appenv
RUN /appenv/bin/pip install twisted pyramid

COPY catx/ /mnt/src/catx/
COPY setup.py /mnt/src/
RUN /appenv/bin/pip install /mnt/src/

RUN tar cvzf /appenv.tar.gz /appenv
```

Better Version 1

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199 \
     as builder

RUN pypy3 -m venv /appenv
RUN /appenv/bin/pip install twisted pyramid

COPY catx/ /mnt/src/catx/
COPY setup.py /mnt/src/
RUN /appenv/bin/pip install /mnt/src/

RUN tar cvzf /appenv.tar.gz /appenv

FROM moshez/pypy3:2017-10-30T09-29-03-882199
COPY --from=builder /appenv.tar.gz /
RUN tar xvzf /appenv.tar.gz
ENTRYPOINT ["/appenv/bin/python", "-m", "twisted",
            "--wsgi=catx.app.app"]
```

Locking

```
$ git checkout -b updating-third-party
$ docker build -t temp-image -f lock.docker .
$ docker run --rm -it temp-image > \
  requirements.locked.txt
$ git commit -a -m 'Update 3rd party'
$ git push
$ # Follow code workflow
```

Locking

```
FROM moshez/pypy3:2017-10-30T09-29-03-882199
COPY lock-requirements requirements.loose.txt /
ENTRYPOINT ["/lock-requirements"]
```

Locking

```
import subprocess, sys
subprocess.check_output([sys.executable, "-m", "ven
                         "/envs/loose"])
subprocess.check_output(["/envs/loose/bin/pip", "in
                         "/requirements.loose.txt"]
frozen = subprocess.check_output(["/envs/loose/bin/
                                  "freeze", "--all"
with open("/requirements.locked.txt", 'wb') as fp:
    fp.write(frozen)
```

Summary

- Use multi-stage builds

Summary

- Use multi-stage builds
- Lock Python requirements

Summary

- Use multi-stage builds
- Lock Python requirements
- Think about your entrypoint