Distroless Docker Containers for Machine Learning at ING

About me

- Bachelor of Computer Science at Delft University
- Currently doing my Master’s in Computer Science
  - Specializing in Data Science
- Working as a machine learning engineer at ING bank
  - Productionalizing Machine Learning
- First time giving a talk (scary!)

What I’ll be talking about today

- Some context: machine learning in production
- A journey of a simple use case
    - Analyzing our use case
    - *Distrofying* our use case

Machine Learning in production

- Many teams, many models
  - Having each team manage their model and exposing an API does not promote uniformity within an organisation

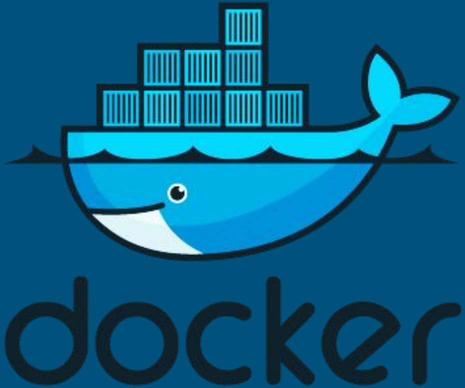
Enter: The Machine Learning Platform

- Many models on one infrastructure
  - ‘Container platform’
- Specialized pipelines for data scientists
- Model orchestration
- Many models running in their own environments
  - Excellent use-case for containers!

Machine Learning, some concerns

- Machine learning models handle sensitive data
  - Combination of features can lead to identification
  - Anonymization is *very* difficult!
- Parameters of a machine learning model may be used maliciously or may also contain sensitive information
  - For example: transforming words into vectors
- **This talk: be aware of the container your model runs in**

Our little model



Our little model

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()

model = RandomForestClassifier()
model.fit(iris.data, iris.target)
```

Our little model, continued

```python
import numpy as np
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json["data"]
    prediction = model.predict(np.expand_dims(data, axis=0))
    return jsonify({"result": int(prediction[0])})
```

Our little model, a quick test

```
$ flask run
```

```
$ curl -H 'Content-Type: application/json' \
    -d '{"data": [5.9, 3.0,  5.1, 1.8]}' \
    -X POST http://localhost:5000/predict
```

Returns...

```
{
    "result": 2
}
```

Our little model, dockerized

```
FROM    python:3
WORKDIR /usr/src/app
COPY    requirements.txt ./
RUN     pip install -r requirements.txt
COPY    app.py app.py
CMD     ["flask", "run"]
```

```
$ docker build -t my-python-app:1.0.0 .
$ docker run -p 5000:5000 --name app my-python-app:1.0.0
```

Our little model, a quick test

```
flask run

curl -H 'Content-Type: application/json' \
    -d '{"data": [5.9, 3.0,  5.1, 1.8]}' \
    -X POST http://localhost:5000/predict
```

Returns...

```
{
    "result": 2
}
```

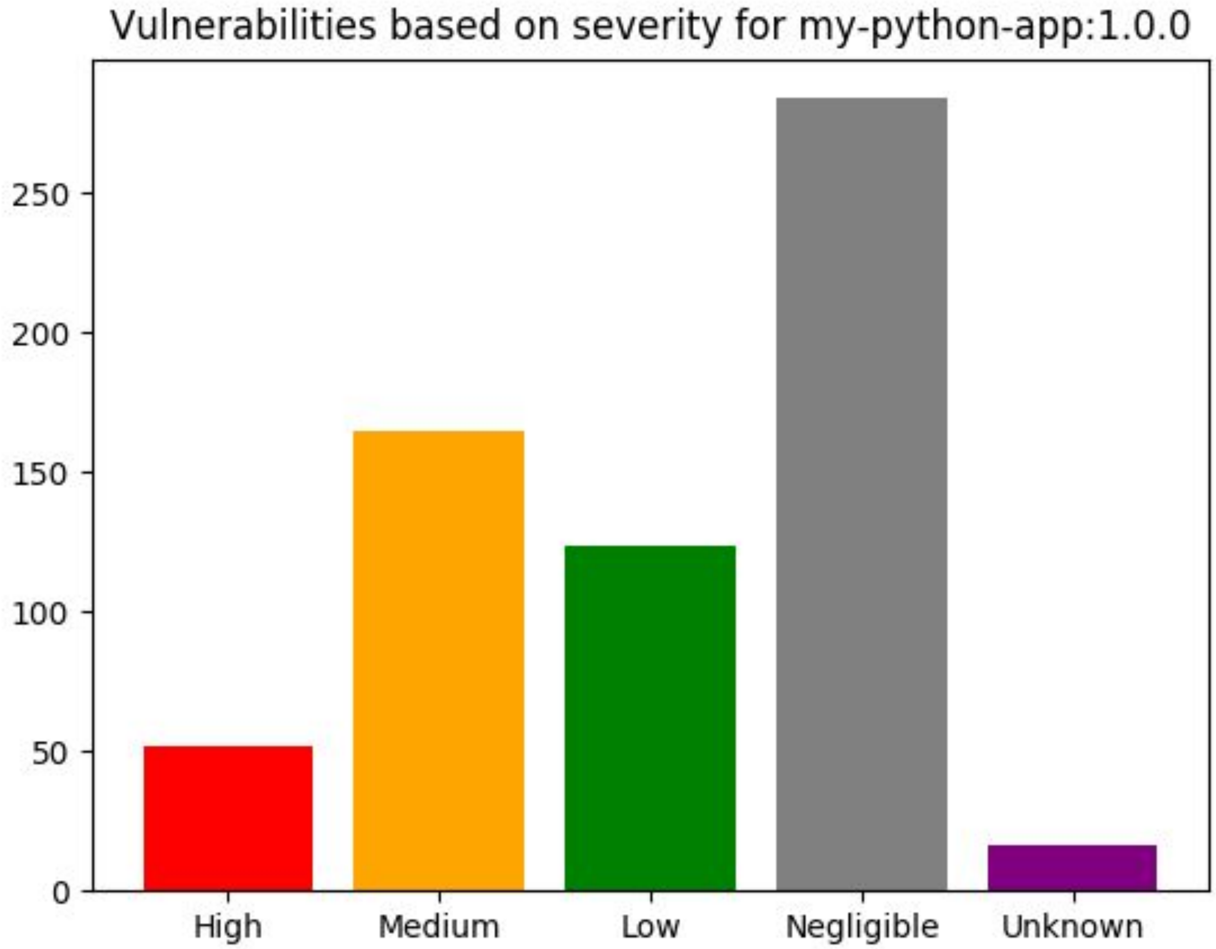
Scanning images

- Dynamic analysis
    - We can actively monitor the running container
- Static analysis
    - We can perform analysis before running the container

Scanning images, static analysis with clair

- Simply specify the image!

```
$ clair-scanner -r report.json --ip docker.for.mac.localhost \
    my-python-app:1.0.0
```


Vulnerabilities based on severity for my-python-app:1.0.0
250
200
150
100
50
0
High
Medium
Low
Negligible
Unknown

Inspecting the image, miscellaneous

- The size of the image is quite large, 1.1 GB
- Any user who is part of the docker group can attach a shell and modify the docker container

```
$ docker exec -it --name app sh
# ls
...
```

Distroless, what is it?

“"Distroless" images contain only your application and its runtime dependencies. They do not contain package managers, shells or any other programs you would expect to find in a standard Linux distribution.”

https://github.com/GoogleContainerTools/distroless

Our little model, revisited

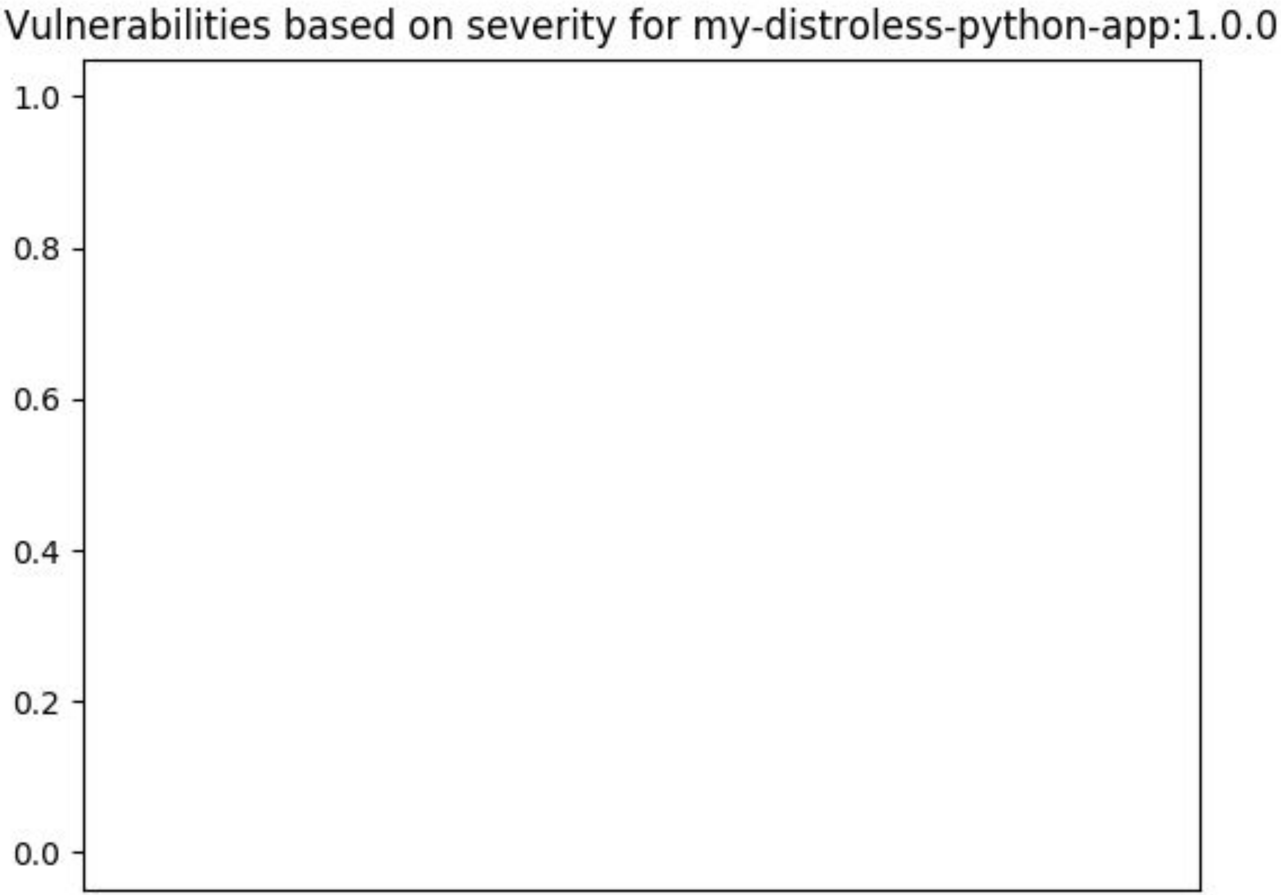
```
FROM    gcr.io/distroless/python3
WORKDIR /usr/src/app
COPY    requirements.txt ./
RUN     pip install -r requirements.txt
COPY    app.py app.py
CMD     ["flask", "run"]
```

```
$ docker build -t my-python-app:1.0.0 .
/bin/sh: 1: pip: not found
```

Our little model, revisited, multi-stage

```
FROM    python:3.5 AS build
COPY    requirements.txt  .
RUN     pip install -r ./requirements.txt

FROM    gcr.io/distroless/python3
COPY    --from=build /usr/local/lib/python3.5/site-packages/ \
                     /usr/lib/python3.5/.
ENV     LC_ALL C.UTF-8
WORKDIR /usr/src/app
COPY    app.py app.py
CMD     ["-m", "flask", "run"]
```


Vulnerabilities based on severity for my-distroless-python-app:1.0.0
1.0
0.8
0.6
0.4
0.2
0.0

Inspecting the image, miscellaneous

- The size of the image is smaller, 250MB, quite a significant reduction!
- Any user who is part of the docker group can attach a shell; however, it is more difficult to modify the docker container

```
docker exec -it --name app sh
# ls
sh: 1: ls: not found
```

But we can do better!

- If we inspect the image, 50MB originates from the distroless image and 200MB from the python dependencies!

A short introduction, PyInstaller

- PyInstaller allows us to freeze our dependencies
    - This way, we can decrease the size of our images significantly!

Our little model, some changes

```
$ flask run
```

```
app = Flask(__name__)


...


if __name__ == "__main__":
    app.run()
```

```
$ python app.py
```

Our little model, with PyInstaller

```
FROM    python:3 AS build
WORKDIR /usr/src/app
COPY    requirements.txt app.py ./
RUN     pip install --upgrade pip --upgrade setuptools && \
            pip install -r requirements.txt && \
            pyinstaller app.py

FROM    gcr.io/distroless/python3
COPY    --from=build /usr/src/app/dist /usr/src/app/dist
ENTRYPOINT  [“/usr/src/app/dist/app”]
```

Our little model, attempt #1

```
$ docker run my-distroless-python-app:1.0.0
ModuleNotFoundError: No module named 'sklearn.utils._cython_blas'
```

- Sometimes we have to help PyInstaller find imports through specification files

Our little model, PyInstaller spec file

```
a = Analysis(['app.py'],
     hiddenimports= [
          'sklearn.utils._cython_blas',
          'sklearn.ensemble',
          'sklearn.neighbors.typedefs',
          'sklearn.neighbors.quad_tree',
          'sklearn.tree._utils'
      ],
     datas=collect_data_files(‘sklearn.datasets’)
)
```

```
...
COPY     requirements.txt \
             app.py app.spec .
...
RUN      pyinstaller app.spec
...
```

Our little model, attempt #2

```
$ docker run my-distroless-python-app:1.0.0
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- The size of the image has been reduced to 97MB!

Our little model, further improvements

- Bundle PyInstaller executable with python library files and use scratch image

Lastly, Some docker tips

- Don’t run as root
- Use image hash instead of image name and tag
- Build your own distroless images
- Sign docker images

To summarize

- Be careful in which images you choose for your models
- Use smaller (distroless) images to limit possible exposure to vulnerabilities

Thanks so much!

- Code highlighter for slides:
    - https://github.com/romannurik/SlidesCodeHighlighter
- Clair-scanner:
    - https://github.com/arminc/clair-scanner
- Awesome libraries used:
    - https://github.com/matplotlib/matplotlib
    - https://github.com/numpy/numpy
    - https://github.com/scikit-learn/scikit-learn
    - https://github.com/pallets/flask
    - https://github.com/docker/docker-ce