ALLGEMEIN, DATA SCIENCE, EVENTS, NEWS, NEWSTICKER

Multi Stage Python

When shipping applications using containers, one often is confronted with overly large final images. Multi-stage builds are a common way to circumvent this issue, especially for compiled languages like Go or Java. In our latest blog post we show how to utilise multi-stage builds for python images to bring down image sizes and thereby improving security.

Using docker multi-stage builds to reduce image size

When writing container images, it is always preferable to have smaller images and to only include what is really needed. This has two main advantages:

- images take up less storage space
- security vulnerabilities of software that is not installed cannot be exploited

Example dockerfile

Let's take a very simple image that starts off with an official python-docker image, using a slim version of the current stable debian release, bullseye. We are going to use python 3.8, but this is not important. The image is pulled from docker hub.

The dockerfiles are provided in this repository in the docker directory and can be built using './bin/build.sh' on Linux.

Reducing the image size

The all-in-one solution

We are simply going to `COPY` over a `requirements.txt` and install it using `pip`, as this is a fairly common use case. I have chosen to install pyodbc since it has some system
dependencies that have to be installed.

Note: The dockerfiles are a minimal reproducible example and do not follow some common best practices!

Unset

```
```dockerfile
FROM python:3.8-slim-bullseye

RUN apt-get update && \
    apt-get install -y gcc g++ unixodbc-dev

COPY requirements.txt /tmp/requirements.txt

RUN pip3 install --no-cache-dir -r /tmp/requirements.txt
```
```

After building the image, we get an image size of 422MB:

Unset

```
```bash
$ docker images original
```
```

```
REPOSITORY   TAG       IMAGE ID       CREATED
SIZE
original     latest    09ec99a1bafa   About a minute ago
422MB
```

Introducing: multi-stage

Multi-stage builds are a neat way to keep build dependencies from blowing up the size of your final image. I am not going to go into detail on how they work.

I am going to create wheels. From the pip documentation:

> *"Wheel is a built-package format, and offers the advantage of not recompiling your software during every install."*

And this is exactly what we want. Compile it and then reuse it without compilation.

The original dockerfile is split in 2: the builder image will install the build dependencies and create the wheels. The final image will install the packages from the wheels without the need to install any build tools:

Unset

```
the wheels. The final image will install the packages from
the wheels without the need to install any
```dockerfile
ARG WHEEL_DIST="/tmp/wheels"

FROM python:3.8-slim-bullseye as builder

ARG WHEEL_DIST
```
```

```
RUN apt-get update && \
    apt-get install -y gcc g++ unixodbc-dev

COPY requirements.txt /tmp/requirements.txt

RUN python3 -m pip wheel -w "${WHEEL_DIST}" -r
/tmp/requirements.txt


FROM python:3.8-slim-bullseye

ARG WHEEL_DIST

COPY --from=builder "${WHEEL_DIST}" "${WHEEL_DIST}"

WORKDIR "${WHEEL_DIST}"

RUN pip3 --no-cache-dir install *.whl
```

The size of the image has gone down significantly:

Unset

```bash
$ docker images multi-stage
REPOSITORY    TAG       IMAGE ID       CREATED
SIZE
multi-stage   latest    b90d97997b3b   44 seconds ago
128MB
```

The difference is starker when considering the size of the base image:

Unset

```bash
docker images python:3.8-slim-bullseye
REPOSITORY   TAG                 IMAGE ID       CREATED
SIZE
python       3.8-slim-bullseye   caf584a25606   5 days ago
122MB
```

Increasing security

Smaller images can contribute to more security.
To illustrate this point, we are going to use trivy to scan our images for known security vulnerabilities.

The base image has 85 vulnerabilities:

- LOW: 12
- MEDIUM: 35
- HIGH: 30
- CRITICAL: 8

The original image that includes the build tools has 331 total vulnerabilities:

- UNKNOWN: 2
- LOW: 24
- MEDIUM: 175
- HIGH: 109
- CRITICAL: 21

The multi-stage image again has 85 images that it "inherits" from the base image, but does not introduce any new ones.

Summary

When using python, do utilise wheels and multi-stage builds to decrease the image size and increase the security of your deployements.