Dockerfiles allow us to build our own images containing any software we need. However, it is important to ensure that our Dockerfiles are built to produce small, efficient images that do not contain unnecessary data. In this lesson, we will briefly discuss some general tips for creating efficient images. We will also demonstrate how to use multi-stage builds to significantly decrease image size in certain situations.

Relevant Documentation

- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- https://docs.docker.com/develop/develop-images/multistage-build/

Lesson Reference

Create some project directories:

```
cd ~/
mkdir efficient
mkdir inefficient
cd inefficient
```

Create the source code file:

```
vi helloworld.go
```

```
package main
import "fmt"
func main() {
    fmt.Println("hello world")
}
```

Create the Dockerfile:

```
vi Dockerfile
```

```
FROM golang:1.12.4
WORKDIR /helloworld
COPY helloworld.go .
RUN GOOS=linux go build -a -installsuffix cgo -o helloworld .
CMD ["./helloworld"]
```

Build and test the `inefficient` image:

```
docker build -t inefficient .
docker run inefficient
docker image ls
```

Switch to the `efficient` project directory and copy the files from the `inefficient` project:

```
cd ~/efficient
cp ../inefficient/helloworld.go ./
```

```
cp ../inefficient/Dockerfile ./
```

Change the Dockerfile to use a multi-stage build:

```
vi Dockerfile
```

```
FROM golang:1.12.4 AS compiler
WORKDIR /helloworld
COPY helloworld.go .
RUN GOOS=linux go build -a -installsuffix cgo -o helloworld .

FROM alpine:3.9.3
WORKDIR /root
COPY --from=compiler /helloworld/helloworld .
CMD ["./helloworld"]
```

Build and test the `efficient` image:

```
docker build -t efficient .
docker run efficient
docker image ls
```