Prerequisites

This tutorial will need the master build of Docker, which is available on Play With Docker.

Multi-staged builds

A common pipe-line for building applications in Docker involves adding SDKs and runtimes, followed by adding code and building it. The most efficient way to get a small image tends to be to use 2-3 Dockerfiles with different filenames where each one takes the output of the last. This is referred to as the *Builder pattern* in the Docker community.

This lab explores a new bleeding-edge feature called Multi-stage builds. It is not yet released into a Docker version, but when it is available on the Docker Hub/Cloud and for all the Docker editions it will mean we can use a single Dockerfile with multiple stages instead of the *Builder pattern*.

Let’s build a simple Golang application which counts internal/external facing anchor tags to help us come up with an SEO rating.

Let’s try out the href-counter Docker image from the hub, then look at how to re-build from the Github repository:

docker run -e url=https://news.ycombinator.com alexellis2/href-counter

{"internal":197,"external":32}

You get a JSON object returned giving the total amount of internal vs external links.

Let’s clone the source:

git clone https://github.com/alexellis/href-counter

cd href-counter

Cloning into 'href-counter'...

remote: Counting objects: 24, done.

remote: Compressing objects: 100% (19/19), done.

remote: Total 24 (delta 7), reused 12 (delta 1), pack-reused 0

Unpacking objects: 100% (24/24), done.

The old way of doing things

Let’s build the Docker image with all the Golang toolchain and see how big the image comes out as:

docker build -t alexellis2/href-counter:sdk . -f Dockerfile.build

Now check the size of the image:

docker images |grep href-counter

docker images |grep href-counter

href-counter sdk 131c782e8c35 30 second

s ago 692MB

The docker history command will show you that the layers we added during the build are only a small part of the resulting image (about 20MB +/-):

docker history alexellis2/href-counter:sdk |head -n 4

IMAGE CREATED CREATED BY

SIZE COMMENT

f8b1953fb9c7 1 second ago /bin/sh -c CGO\_ENABLED=0 GOOS=linux go bui... 5.64MB

5d24895500e8 9 seconds ago /bin/sh -c #(nop) COPY file:d3eec1f1fefbec... 1.71kB

d83dc0785057 9 seconds ago /bin/sh -c go get -d -v golang.org/x/net/html 13.6MB

c6f59b210906 11 seconds ago /bin/sh -c #(nop) WORKDIR /go/src/github.c... 0B

The image is quite large, but this Golang package can be built into a very small binary with no external dependencies, then added to an Alpine Linux base image.

The builder pattern

Type in cat builder.sh so you can see how the builder pattern uses two separate Dockerfiles. This will help us get the context for the next step where we will use a single Dockerfile.

./build.sh

*#!/bin/sh*

echo Building alexellis2/href-counter:build

docker build -t alexellis2/href-counter:build . -f Dockerfile.build

docker create --name extract alexellis2/href-counter:build

docker cp extract:/go/src/github.com/alexellis/href-counter/app ./app

docker rm -f extract

echo Building alexellis2/href-counter:latest

docker build -t alexellis2/href-counter:latest .

As you can see there are quite a few intermediate steps required to create an optimized image using the *Builder pattern*.

Let’s see how big the Docker image came out as:

docker images |grep alexellis2/href-counter

This is much smaller than when we built our first image with the Golang SDK included.

Multi-stage build example

While the builder pattern helps us achieve a small image, it does require extra leg-work for every piece of software we want to package in Docker.

Here is where multi-stage builds help us out. Instead of using a shell script to orchestrate two separate Dockerfiles, we can just use one and define stages throughout.

To use the Dockerfile.multi file in the Github repository to do a multi-stage build, then check the size of the resulting image against that of the image created by the Golang SDK base and the builder pattern.

docker build -t alexellis2/href-counter:multi . -f Dockerfile.multi

docker images |grep href-counter

alexellis2/href-counter multi 44852229a1cc 2 minutes ago 10.3MB

alexellis2/href-counter latest bb997f819fbb 3 minutes ago 10.3MB

alexellis2/href-counter build 298d3b970412 4 minutes ago 692MB

alexellis2/href-counter sdk 298d3b970412 4 minutes ago 692MB

alexellis2/href-counter <none> b0a73b688243 13 days ago 11.6MB

Here is an example of building “Hello World” in Go.

Create a new folder:

mkdir hello

cd hello

Create the app.go file:

echo 'package main

import "fmt"

func main() {

fmt.Println("Hello world!")

}

' | tee app.go

Create a Dockerfile with the following contents:

echo '

FROM golang:1.7.3

COPY app.go .

RUN go build -o app app.go

FROM scratch

COPY --from=0 /go/app .

CMD ["./app"]

' | tee Dockerfile

Now build and run the Dockerfile:

docker build -t hello-world-lab .

docker run hello-world-lab

The resulting size of hello-world is very small:

docker images |grep hello-world-lab

More about the lab

This lab was built from a [blog post](http://blog.alexellis.io/mutli-stage-docker-builds/). by [Alex Ellis](https://twitter.com/alexellisuk)

* Blog: [Builder pattern vs. Multi-stage builds in Docker](http://blog.alexellis.io/mutli-stage-docker-builds/).

Note: We do not recommend moving over to multi-stage builds until they are fully available on the Docker Hub/Cloud and all editions of Docker. The example in build.sh provides an interim solution for using separate Dockerfiles to build and ship code.