DOCKERFILES : BUILDING DOCKER IMAGES AUTOMATICALLY II - REVISITING FROM, MAINTAINER, BUILD CONTEXT, AND CACHING

Continued from ...

COntinued from Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context (http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically.php).

This chapter is similar to the previous one, Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context (/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically.php), that's because we want to make sure how **docker build** works with **context**.

Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

Docker & K8s

Docker install on Amazon Linux AMI (/DevOps/Docker/Docker_Install_C

Docker install on EC2 Ubuntu

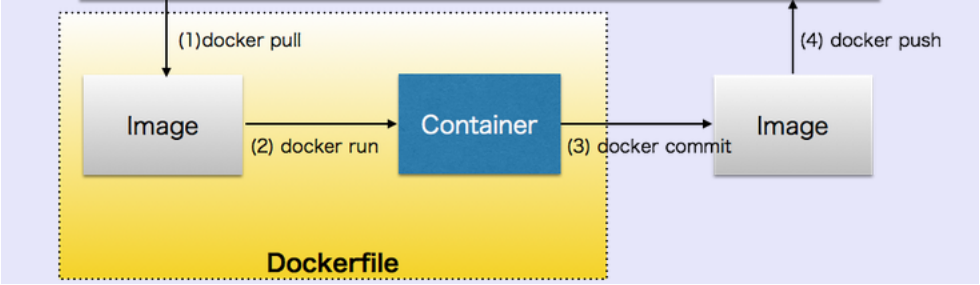

Image source: Docker (http://hondou.homedns.org/pukiwiki/pukiwiki.php?Docker%20%A4%B6%A4%C3%A4%AF%A4%EA%B8%C0%A4%A6%A4%C8%B2%BF%3F)

In this chapter, we're going to learn more on how to automate this process via instructions in Dockerfiles.

Base image

Let's download our base image:

```
$ docker pull debian:latest
debian:latest: The image you are pulling has been verified
511136ea3c5a: Pull complete
f10807909bc5: Pull complete
f6fab3b798be: Pull complete
Status: Downloaded newer image for debian:latest

$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED        VIRTUAL SIZE
debian          latest    f6fab3b798be   2 weeks ago    85.1 MB
```

In our local working directory, we have only one file, `Dockerfile`:

```
$ ls
Dockerfile
```

Each `Dockerfile` is a script, composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image in order to create (or form) a new one. They are used for organizing things and greatly help with deployments by simplifying the process start-to-finish.

14.04 (/DevOps/Docker/Docker_Install_C

Docker container vs Virtual Machine (/DevOps/Docker/Docker_Containe

Docker install on Ubuntu 14.04 (/DevOps/Docker/Docker_Install_C

Docker Hello World Application (/DevOps/Docker/Docker_Hello_W

Nginx image - share/copy files, Dockerfile (/DevOps/Docker/Docker_Nginx_W

Working with Docker images : brief introduction (/DevOps/Docker/Docker_Working

Docker image and container via docker commands (search, pull, run, ps, restart, attach, and rm) (/DevOps/Docker/Docker_Commar

More on docker run command (docker run -it, docker run --rm, etc.) (/DevOps/Docker/Docker_Run_Cor

Docker Networks - Bridge Driver Network (/DevOps/Docker/Docker-Bridge-Driver-Networks.php)

Docker Persistent Storage (/DevOps/Docker/Docker_Containe

File sharing between host and container (docker run -d -p -v) (/DevOps/Docker/Docker_File_Sha

Linking containers and volume for datastore (/DevOps/Docker/Docker_Containe

Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context (/DevOps/Docker/Docker_Dockerfi

Dockerfile - Build Docker images automatically II - revisiting FROM, MAINTAINER, build context, and caching (/DevOps/Docker/Docker_Dockerfi

Dockerfile - Build Docker images automatically III - RUN (/DevOps/Docker/Docker_Dockerfi

Dockerfile - Build Docker images

docker build 'FROM'

Let's look at the syntax of `docker build` command:

```
$ docker build --help

Usage: docker build [OPTIONS] PATH | URL | -

Build a new image from the source code at PATH

  -t, --tag=""         Repository name (and optionally a tag) to be applied to the resulting imag
```

`Dockerfiles` begin with defining an image `FROM` which the build process starts. Followed by various other methods, commands and arguments (or conditions), in return, provide a new image which is to be used for creating docker containers.

```
FROM debian:latest
MAINTAINER devops@bogotobogo.com
```

Let's run `docker build` command with the two-line `Dockerfile`:

```
$ docker image build -t bogodevops/demo:v1 .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM debian:latest
 ---> f6fab3b798be
Step 1 : MAINTAINER k@bogotobogo.com
 ---> Running in 4181b54ab22e
 ---> 511bcbdd59ba
Removing intermediate container 4181b54ab22e
Successfully built 511bcbdd59ba
```

Now if we list the images:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              VIRTUAL SIZE
bogodevops/demo     v1                  511bcbdd59ba        About a minute ago   85.1 MB
debian              latest              f6fab3b798be        2 weeks ago          85.1 MB
```

Note that the path to the source repository defines where to find the context of the build. The build is run by the Docker daemon, not by the CLI, so the whole context must be transferred to the daemon. The Docker CLI reports "Sending build context to Docker daemon" when the context (2.56kB) is sent to the daemon as shown in the output:

```
Sending build context to Docker daemon  2.56 kB
```

If we send big chuck to the daemon, it will take longer to copy things. For example, if we send duplicate device files(/de/zero) with `dd`:

automatically IV - CMD (/DevOps/Docker/Docker_Dockerfi

Dockerfile - Build Docker images automatically V - WORKDIR, ENV, ADD, and ENTRYPOINT (/DevOps/Docker/Docker_Dockerfi

Docker - Apache Tomcat (/DevOps/Docker/Docker_Apache_

Docker - NodeJS (/DevOps/Docker/Docker-NodeJS.php)

Docker - NodeJS with hostname (/DevOps/Docker/Docker-NodeJS-with-hostname.php)

Docker Compose - NodeJS with MongoDB (/DevOps/Docker/Docker-Compose-Node-MongoDB.php)

Docker - Prometheus and Grafana with Docker-compose (/DevOps/Docker/Docker_Prometh

Docker - StatsD/Graphite/Grafana (/DevOps/Docker/Docker_StatsD_C

Docker - Deploying a Java EE JBoss/WildFly Application on AWS Elastic Beanstalk Using Docker Containers (/DevOps/Docker/Docker_Containe

Docker : NodeJS with GCP Kubernetes Engine (/DevOps/Docker/Docker-NodeJS-GCP-Kubernetes-Engine.php)

Docker : Jenkins Multibranch Pipeline with Jenkinsfile and Github (/DevOps/Docker/Docker-Jenkins-Multibranch-Pipeline-with-Jenkinsfile-and-Github.php)

Docker : Jenkins Master and Slave (/DevOps/Docker/Docker-Jenkins-Master-Slave-Agent-ssh.php)

Docker - ELK : ElasticSearch, Logstash, and Kibana (/DevOps/Docker/Docker_ELK_Elas

Docker - ELK 7.6 : Elasticsearch on Centos 7 (/DevOps/Docker/Docker_ELK_7_6_

Docker - ELK 7.6 : Filebeat on

```
$ dd if=/dev/zero of=testimage bs=4096 count=8192
8192+0 records in
8192+0 records out
33554432 bytes (34 MB) copied, 0.118561 s, 283 MB/s

$ ls
Dockerfile  testimage

$ docker image build -t bogodevops/demo:v1 .
Sending build context to Docker daemon 33.56 MB
Sending build context to Docker daemon
Step 0 : FROM debian:latest
 ---> f6fab3b798be
Step 1 : MAINTAINER k@bogotobogo.com
 ---> Using cache
 ---> 511bcbdd59ba
Successfully built 511bcbdd59ba
```


Note that the size has been increased from 2.56kb to 33.56MB. That's why the Docker document (http://docs.docker.com/reference/builder/) gives us a Warning like this:

> "Warning Avoid using your root directory, /, as the root of the source repository. The docker build command will use whatever dicrectory contains the Dockerfile as the build context (including all of its subdirectories). The build context will be sent to the Docker daemon before building the image, which means if you use / as the source repository, the entire contents of your hard drive will get sent to the daemon (and thus to the machine running the daemon). You probably don't want that."

Or like this:

> "Warning: Do not use your root directory, /, as the PATH as it causes the build to transfer the entire contents of your hard drive to the Docker daemon.

So, we should be aware of the context of our build directory!

Again:

Centos 7 (/DevOps/Docker/Docker_ELK_7_6_

Docker - ELK 7.6 : Logstash on Centos 7 (/DevOps/Docker/Docker_ELK_7_6_

Docker - ELK 7.6 : Kibana on Centos 7 Part 1 (/DevOps/Docker/Docker_ELK_7_6_

Docker - ELK 7.6 : Kibana on Centos 7 Part 2 (/DevOps/Docker/Docker_ELK_7_6_

Docker - ELK 7.6 : Elastic Stack with Docker Compose (/DevOps/Docker/Docker_ELK_7_6_

Docker - Deploy Elastic Cloud on Kubernetes (ECK) via Elasticsearch operator on minikube (/DevOps/Docker/Docker_Kuberne

Docker - Deploy Elastic Stack via Helm on minikube (/DevOps/Docker/Docker_Kuberne

Docker Compose - A gentle introduction with WordPress (/DevOps/Docker/Docker-Compose.php)

Docker Compose - MySQL (/DevOps/Docker/Docker-Compose-MySQL.php)

MEAN Stack app on Docker containers : micro services (/MEAN-Stack/MEAN-Stack-NodeJS-Angular-Docker.php)

Docker Compose - Hashicorp's Vault and Consul Part A (install vault, unsealing, static secrets, and policies) (/DevOps/Docker/Docker-Vault-Consul.php)

Docker Compose - Hashicorp's Vault and Consul Part B (EaaS, dynamic secrets, leases, and revocation) (/DevOps/Docker/Docker-Vault-Consul-B.php)

Docker Compose - Hashicorp's Vault and Consul Part C (Consul) (/DevOps/Docker/Docker-Vault-Consul-C.php)

Docker Compose with two

> "The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it's best to start with an empty directory as **context** and keep your **Dockerfile** in that directory. Add only the files needed for building the **Dockerfile**"

Also note the the difference in the Step 1 of the two cases:

In the first instance of `docker build`:

```
Step 1 : MAINTAINER k@bogotobogo.com
 ---> Running in 4181b54ab22e
 ---> 511bcbdd59ba
```

But in the second run, Docker used `cache`:

```
Step 1 : MAINTAINER k@bogotobogo.com
 ---> Using cache
 ---> 511bcbdd59ba
```

When we build a Docker image, it's using a Dockerfile, and every instruction in the Dockerfile is run inside of a container. If that returns successfully, then that container is stored as a new image.

In our case, in Step 0, we created 'f6fab3b798be' which is a hash identifier, and Step 1, we created '511bcbdd59ba' hash.

Note that in our 2nd run (the 'docker run' with 'dd'), the hash is the same. What does this mean? If the instructions in our Dockerfile are the same, Docker uses the cache:

```
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
bogodevops/demo     v1                  511bcbdd59ba        57 minutes ago      85.1 MB
debian              latest              f6fab3b798be        2 weeks ago         85.1 MB
<none>              <none>              f10807909bc5        2 weeks ago         85.1 MB
<none>              <none>              511136ea3c5a        17 months ago       0 B
```

So, every step along the way, we create a new image. As it succeeds, we'll build a new layer on top of the previous one as we read in an instruction. As this caching allows us to build other environment similar to the previous image without rebuilding from every steps involved.

containers - Flask REST API service container and an Apache server container (/DevOps/Docker/Docker-Compose-FlaskREST-Service-Container-and-Apache-Container.php)

Docker compose : Nginx reverse proxy with multiple containers (/DevOps/Docker/Docker-Compose-Nginx-Reverse-Proxy-Multiple-Containers.php)

Docker compose : Nginx reverse proxy with multiple containers (/DevOps/Docker/Docker-Compose-Nginx-Reverse-Proxy-Multiple-Containers.php)

Docker & Kubernetes : Envoy - Getting started (/DevOps/Docker/Docker-Envoy-Getting-Started.php)

Docker & Kubernetes : Envoy - Front Proxy (/DevOps/Docker/Docker-Envoy-Front-Proxy.php)

Docker & Kubernetes : Ambassador - Envoy API Gateway on Kubernetes (/DevOps/Docker/Docker-Envoy-Ambassador-API-Gateway-for-Kubernetes.php)

Docker Packer (/DevOps/Docker/Docker-Packer.php)

Docker Cheat Sheet (/DevOps/Docker/Docker-Cheat-Sheet.php)

Docker Q & A (/DevOps/Docker/Docker_Q_and_A

Kubernetes Q & A - Part I (/DevOps/Docker/Docker_Kuberne

Kubernetes Q & A - Part II (/DevOps/Docker/Docker_Kuberne

Docker - Run a React app in a docker (/DevOps/Docker/Docker-React-App.php)

Docker - Run a React app in a docker II (snapshot app with nginx) (/DevOps/Docker/Docker-React-App-2-SnapShot.php)