DOCKERFILES : BUILDING DOCKER IMAGES AUTOMATICALLY V - WORKDIR, ENV, ADD, AND ENTRYPOINT

Continued from ...

Continued from Dockerfile - Build Docker images automatically IV - CMD
(http://www.bogotobogo.com/DevOps/Docker/Docker_Dockerfile_to_build_images_automatically_4_CMD.php)

In this chapter, we're going to learn more on how to automate this process via instructions in Dockerfiles. We'll be focusing on WORKDIR, ENV, ADD, and ENTRYPOINT.

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

Dockerfie - WORKDIR & ENV

This section is from http://docs.docker.com/reference/builder/ (http://docs.docker.com/reference/builder/).

```
WORKDIR /path/to/workdir
```

The `WORKDIR` instruction sets the working directory for any `RUN`, `CMD` and `ENTRYPOINT` instructions that follow it in the `Dockerfile`.

It can be used multiple times in the one `Dockerfile`. If a relative path is provided, it will be relative to the path of the previous `WORKDIR` instruction. For example:

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

The output of the final pwd command in this `Dockerfile` would be /a/b/c.

The `WORKDIR` instruction can resolve environment variables previously set using `ENV`. We can only use environment variables explicitly set in the `Dockerfile`. For example:

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
```

The output of the final `pwd` command in this `Dockerfile` would be `/path/$DIRNAME`.

```
ENV <key> <value>
```

The `ENV` instruction sets the environment variable <key> to the value <value>. This value will be passed to all future RUN instructions. This is functionally equivalent to prefixing the command with <key>=<value>

The environment variables set using `ENV` will persist when a container is run from the resulting image. We can view the values using `docker inspect`, and change them using `docker run --env <key>=<value>`.

Note: One example where this can cause unexpected consequences, is setting `ENV DEBIAN_FRONTEND` noninteractive. Which will persist when the container is run interactively; for example:
`docker run -t -i image bash`.

WORKDIR & ENV - sample

Here is our updated `Dockerfile` :

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

# 1 - RUN
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
RUN apt-get clean

# 2 - CMD
#CMD ["htop"]
#CMD ["ls", "-l"]

# 3 - WORKDIR and ENV
WORKDIR /root
ENV DZ version1
```

Let's build the image:

```
$ docker image build -t bogodevops/demo .
Sending build context to Docker daemon  3.072kB
Step 1/7 : FROM debian:latest
 ---> be2868bebaba
Step 2/7 : MAINTAINER k@bogotobogo.com
 ---> Using cache
 ---> e2eef476b3fd
Step 3/7 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
 ---> Using cache
 ---> 32fd044c1356
Step 4/7 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
 ---> Using cache
 ---> 0a5b514a209e
Step 5/7 : RUN apt-get clean
 ---> Using cache
 ---> 5d1578a47c17
Step 6/7 : WORKDIR /root
 ---> Using cache
 ---> 6b1c70e87675
Step 7/7 : ENV DZ version1
 ---> Using cache
 ---> cd195168c5c7
Successfully built cd195168c5c7
Successfully tagged bogodevops/demo:latest
```

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

Here we're using repository name (tag) for the image, and the dot('.') indicates our Dockerfile is in local directory.

What images do we have now?

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              VIRTUAL SIZE
bogodevops/demo     latest              6f9de0a5099f        About a minute ago   96.16 MB
<none>              <none>              d2f3de97b6ef        About an hour ago    96.16 MB
<none>              <none>              e171cd1dd9e7        About an hour ago    96.16 MB
<none>              <none>              b64547129d16        About an hour ago    96.16 MB
bogodevops/demo     v2                  358b5cc4b9fa        2 hours ago          96.16 MB
bogodevops/demo     v1                  511bcbdd59ba        7 hours ago          85.1 MB
debian              latest              f6fab3b798be        2 weeks ago          85.1 MB
```

Note the images tagged with <none>. These are the images which had no tag, and left behind when a new image is tagged as 'latest'.

Now we're going to run a new container and run `bash` inside of it:

```
$ docker container run -it --rm bogodevops/demo /bin/bash
```

-t bogodevops/demo .

We can check the `WORKDIR` and `ENV` settings in our `Dockerfile`:

```
root@52a10702207c:~# pwd
/root
root@52a10702207c:~# echo $DZ
version1
root@52a10702207c:~# exit
exit
```

OK. We've got what we expected.

Dockerfie - ADD

Note: use `COPY` over `ADD`!

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

```
ADD <src>... <dest>
```

The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the container at the path <dest>.

Multiple <src> resource may be specified but if they are files or directories then they must be relative to the source directory that is being built (the context of the build).

Each <src> may contain wildcards and matching will be done using Go's `filepath.Match` rules. For most command line uses this should act as expected, for example:

```
ADD hom* /mydir/        # adds all files starting with "hom"
ADD hom?.txt /mydir/    # ? is replaced with any single character
```

The <dest> is the absolute path to which the source will be copied inside the destination container.

Here is our new `Dockerfile` :

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

# 1 - RUN
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
RUN apt-get clean

# 2 - CMD
#CMD ["htop"]
#CMD ["ls", "-l"]

# 3 - WORKDIR and ENV
WORKDIR /root
ENV DZ version1

# 4 - ADD
ADD run.sh /root/run.sh
RUN chmod +x run.sh
CMD ["./run.sh"]
```

The `run.sh` should be referencing current working directory in our local machine.

Here is the `run.sh` script:

```
#!/bin/sh

echo "The current directory : $(pwd)"
echo "The DZ variable : $DZ"
echo "There are $# arguments: $@"
```

We should build the image:

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)

```
$ docker image build -t bogodevops/demo .
Sending build context to Docker daemon  3.072kB
Step 1/10 : FROM debian:latest
 ---> be2868bebaba
Step 2/10 : MAINTAINER k@bogotobogo.com
 ---> Using cache
 ---> e2eef476b3fd
Step 3/10 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
 ---> Using cache
 ---> 32fd044c1356
Step 4/10 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
 ---> Using cache
 ---> 0a5b514a209e
Step 5/10 : RUN apt-get clean
 ---> Using cache
 ---> 5d1578a47c17
Step 6/10 : WORKDIR /root
 ---> Using cache
 ---> 6b1c70e87675
Step 7/10 : ENV DZ version1
 ---> Using cache
 ---> cd195168c5c7
Step 8/10 : ADD run.sh /root/run.sh
 ---> Using cache
 ---> 4e7d36a09663
Step 9/10 : RUN chmod +x run.sh
 ---> Running in 38cc3f6aface
Removing intermediate container 38cc3f6aface
 ---> a6647d782620
Step 10/10 : CMD ["./run.sh"]
 ---> Running in 5072926ac27a
Removing intermediate container 5072926ac27a
 ---> 009775eea12d
Successfully built 009775eea12d
Successfully tagged bogodevops/demo:latest
```

Then, run a container with no command:

```
$ docker container run -it --rm bogodevops/demo
The current directory : /root
The DZ variable : version1
There are 0 arguments:
```

If we add a command to `docker run`, we get this:

```
$ docker container run -it --rm bogodevops/demo ./run.sh Hello bogotobogo
The current directory : /root
The DZ variable : version1
There are 2 arguments: Hello bogotobogo
```


Docker - NodeJS and MySQL app with React in a docker (/DevOps/Docker/Docker-React-Node-MySQL-App.php)

Docker - Step by Step NodeJS and MySQL app with React - I (/DevOps/Docker/Step-by-Step-React-Node-MySQL-App.php)

Installing LAMP via puppet on Docker (/DevOps/Docker/Installing-LAMP-with-puppet-on-Docker.php)

Docker install via Puppet (/DevOps/Docker/Docker_puppet.p

Nginx Docker install via Ansible (/DevOps/Ansible/Ansible-Deploy-Nginx-to-Docker.php)

Apache Hadoop CDH 5.8 Install with QuickStarts Docker (/Hadoop/BigData_hadoop_CDH5.8

Docker - Deploying Flask app to ECS (/DevOps/Docker/Docker-Flask-ALB-ECS.php)

Docker Compose - Deploying WordPress to AWS (/DevOps/Docker/Docker-Compose-WordPress-AWS.php)

Docker - WordPress Deploy to ECS with Docker-Compose (ECS-CLI EC2 type) (/DevOps/Docker/Docker-ECS-CLI-Docker-Compose-Wordpress-EC2-Type.php)

Docker - AWS ECS service discovery with Flask and Redis (/DevOps/Docker/Docker-ALB-ECS-Fargate.php)

Docker - ECS Fargate (/DevOps/Docker/Docker-ECS-Service-Dicsovery-Redis-Flask.php)

Docker & Kubernetes 1 : minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes 2 : minikube Django with Postgres - persistent volume (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes 3 :

Dockerfie - ENTRYPOINT

`ENTRYPOINT` has two forms:

1. `ENTRYPOINT` ["executable", "param1", "param2"] (the preferred exec form - json array form)
2. `ENTRYPOINT` command param1 param2 (shell form)

An `ENTRYPOINT` allows us to configure a container that will run as an executable.

Any command line arguments passed to `docker run <image>` will be appended to the entrypoint command, and will override all elements specified using CMD. For example, `docker run <image> bash` will add the command argument **bash** to the end of the entrypoint.

Command line arguments to `docker run <image>` will be appended after all elements in an exec form `ENTRYPOINT`, and will override all elements specified using `CMD`. This allows arguments to be passed to the entry point, i.e., `docker run <image> -d` will pass the `-d` argument to the entry point. We can override the `ENTRYPOINT` instruction using the `docker run --entrypoint` flag.

Here is our updated `Dockerfile` which includes `ENTRYPOINT`:

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

# 1 - RUN
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
RUN apt-get clean

# 2 - CMD
#CMD ["htop"]
#CMD ["ls", "-l"]

# 3 - WORKDIR and ENV
WORKDIR /root
ENV DZ version1

# 4 - ADD
ADD run.sh /root/run.sh
RUN chmod +x run.sh
#CMD ["./run.sh"]

# 5 - ENTRYPOINT (vs CMD)
ENTRYPOINT ["./run.sh"]
CMD ["arg1"]
```

minikube Django with Redis and Celery (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes 4 : Django with RDS via AWS Kops (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Kops on AWS (/DevOps/DevOps-Kubernetes-II-kops-on-AWS.php)

Docker & Kubernetes : Ingress controller on AWS with Kops (/DevOps/Docker/Docker-Kubernetes-kops-on-AWS-Ingress.php)

Docker & Kubernetes : HashiCorp's Vault and Consul on minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : HashiCorp's Vault and Consul - Auto-unseal using Transit Secrets Engine (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Persistent Volumes & Persistent Volumes Claims - hostPath and annotations (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Persistent Volumes - Dynamic volume provisioning (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : DaemonSet (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Secrets (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : kubectl command (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Assign a Kubernetes Pod to a particular node in a Kubernetes cluster (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Configure a Pod to Use a ConfigMap (/DevOps/Docker/Docker_Kuberne

AWS : EKS (Elastic Container Service for Kubernetes) (/DevOps/AWS/aws-EKS-Elastic-

Build our image again:

```
$ docker image build -t bogodevops/demo .
Sending build context to Docker daemon  3.072kB
Step 1/11 : FROM debian:latest
 ---> be2868bebaba
Step 2/11 : MAINTAINER k@bogotobogo.com
 ---> Using cache
 ---> e2eef476b3fd
Step 3/11 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
 ---> Using cache
 ---> 32fd044c1356
Step 4/11 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop
 ---> Using cache
 ---> 0a5b514a209e
Step 5/11 : RUN apt-get clean
 ---> Using cache
 ---> 5d1578a47c17
Step 6/11 : WORKDIR /root
 ---> Using cache
 ---> 6b1c70e87675
Step 7/11 : ENV DZ version1
 ---> Using cache
 ---> cd195168c5c7
Step 8/11 : ADD run.sh /root/run.sh
 ---> Using cache
 ---> 4e7d36a09663
Step 9/11 : RUN chmod +x run.sh
 ---> Using cache
 ---> a6647d782620
Step 10/11 : ENTRYPOINT ["./run.sh"]
 ---> Using cache
 ---> 9bb552df306a
Step 11/11 : CMD ["arg1"]
 ---> Using cache
 ---> 7207257fbfc2
Successfully built 7207257fbfc2
Successfully tagged bogodevops/demo:latest
```

Container run without any argument:

```
$ docker container run -it --rm bogodevops/demo
The current directory : /root
The DZ variable : version1
There are 1 arguments: arg1
```

It still runs `run.sh` shell. If we pass in something like `/bin/bash` :

```
$ docker run -it --rm bogodevops/demo /bin/bash
The current directory : /root
The DZ variable : version1
There are 1 arguments: /bin/bash
```

Still it runs `run.sh` file while `/bin/bash` was passed in as an argument.

Container-Service-Kubernetes.php)

Docker & Kubernetes : Run a React app in a minikube (/DevOps/Docker/Docker-Kubernetes-React-App.php)

Docker & Kubernetes : Minikube install on AWS EC2 (/DevOps/Docker/Docker-Kubernetes-Minikube-install-on-AWS-EC2.php)

Docker & Kubernetes : Cassandra with a StatefulSet (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Terraform and AWS EKS (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Pods and Service definitions (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Headless service and discovering pods (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Service IP and the Service Type (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Kubernetes DNS with Pods and Services (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes - Scaling and Updating application (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Horizontal pod autoscaler on minikubes (/DevOps/Docker/Docker-Kubernetes-Horizontal-Pod-Autoscaler.php)

Docker & Kubernetes : NodePort vs LoadBalancer vs Ingress (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Load Testing with Locust on GCP Kubernetes (/DevOps/Docker/Docker-Load-Testing-with-Locust-on-GCP-Kubernetes.php)

Docker & Kubernetes : From a monolithic app to micro services

ENTRYPOINT vs CMD

When we run an Ubuntu image, it exits immediately as we can see below:

```
$ docker run ubuntu:18.04
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
6cf436f81810: Pull complete
987088a85b96: Pull complete
b4624b3efe06: Pull complete
d42beb8ded59: Pull complete
Digest: sha256:7a47ccc3bbe8a451b500d2b53104868b46d60ee8f5b35a24b41a86077c650210
Status: Downloaded newer image for ubuntu:18.04

$ docker ps -a
CONTAINER ID IMAGE         COMMAND      CREATED         STATUS                     PORTS
9ba1aa158caf ubuntu:18.04 "/bin/bash" 11 seconds ago Exited (0) 10 seconds ago
```

Why is that? Why it exited?

Unlike VMs which are meant to host OS, containers are meant to run a task or a process such as a web server/application or a db. So, once a task is complete, a container exits. A container lives as long as a process within it is running. If an application in a container crashes, container exits.

So, who defines which process should be running inside a container?

Let's look into the following Dockerfile for nginx (), specially the **CMD[]** instruction:

on GCP Kubernetes (/DevOps/Docker/Docker-from-Monolithic-to-Micro-services-GCP-Kubernetes.php)

Docker & Kubernetes : Rolling updates (/DevOps/Docker/Docker-Kubernetes-Rolling-Updates.php)

Docker & Kubernetes : Deployments to GKE (Rolling update, Canary and Blue-green deployments) (/DevOps/Docker/Docker-Rolling-Update-Canary-Blue-Green-Deployments-to-GKE-Kubernetes.php)

Docker & Kubernetes : Slack Chat Bot with NodeJS on GCP Kubernetes (/DevOps/Docker/Docker-Slack-NodeJS-ChatBot-GCP-Kubernetes.php)

Docker & Kubernetes : Continuous Delivery with Jenkins Multibranch Pipeline for Dev, Canary, and Production Environments on GCP Kubernetes (/DevOps/Docker/Docker-Continuous-Delivery-with-Jenkins-Multibranch-Pipeline-for-Dev-Canary-Production-Environments-GCP-Kubernetes-Engine-Namespace.php)

Docker & Kubernetes - MongoDB with StatefulSets on GCP Kubernetes Engine (/DevOps/Docker/Docker_MongoD

Docker & Kubernetes : Nginx Ingress Controller on minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Nginx Ingress Controller for Dashboard service on Minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Nginx Ingress Controller on GCP Kubernetes (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Kubernetes Ingress with AWS ALB Ingress Controller in EKS (/DevOps/Docker/Docker-Kubernetes-ALB-Ingress-

```
#
# Nginx Dockerfile
#
# https://github.com/dockerfile/nginx
#

# Pull base image.
FROM dockerfile/ubuntu

# Install Nginx.
RUN \
  add-apt-repository -y ppa:nginx/stable && \
  apt-get update && \
  apt-get install -y nginx && \
  rm -rf /var/lib/apt/lists/* && \
  echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
  chown -R www-data:www-data /var/lib/nginx

# Define mountable directories.
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/log/nginx", "/

# Define working directory.
WORKDIR /etc/nginx

# Define default command.
CMD ["nginx"]

# Expose ports.
EXPOSE 80
EXPOSE 443
```

Yes, the **CMD[]** tells the Docker which program should be run when the container starts. In our case, it is the "nginx" command.

For **mysql** Dockerfile it is **mysqld** command:

```
COPY docker-entrypoint.sh /entrypoint.sh
COPY healthcheck.sh /healthcheck.sh
ENTRYPOINT ["/entrypoint.sh"]
HEALTHCHECK CMD /healthcheck.sh
EXPOSE 3306 33060
CMD ["mysqld"]
```

How about our Ubuntu image Dockerfile?

```
...
CMD ["/bin/bash"]
```

It uses **bash** for its default command.

Unlike the web server or a db, the **bash** is not a process, it's just a shell listening and waiting for an input. If it does not get any from a terminal, it exits.

Earlier, when we run a container from the Ubuntu image, it launches a "bash" program but the Docker, by default, not attaching any terminal to a container when it runs. So, the container could not find a terminal, and just exited.

Controller-with-EKS.php)

Docker & Kubernetes : MongoDB / MongoExpress on Minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Setting up a private cluster on GCP Kubernetes (/DevOps/Docker/Docker-setting-up-private-cluster-on-GCP-Kubernetes.php)

Docker & Kubernetes : Kubernetes Namespaces (default, kube-public, kube-system) and switching namespaces (kubens) (/DevOps/Docker/Docker-Kubernetes-Namespaces.php)

Docker & Kubernetes : StatefulSets on minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : StatefulSets on minikube (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : RBAC (/DevOps/Docker/Docker-Kubernetes-RBAC.php)

Docker & Kubernetes Service Account, RBAC, and IAM (/DevOps/Docker/Docker-Kubernetes-Service-Account.php)

Docker & Kubernetes - Kubernetes Service Account, RBAC, IAM with EKS ALB, Part 1 (/DevOps/Docker/Docker-Kubernetes-ALB-on-EKS-1.php)

Docker & Kubernetes : Helm Chart (/DevOps/Docker/Docker_Helm_Ch

Docker & Kubernetes : My first Helm deploy (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Readiness and Liveness Probes (/DevOps/Docker/Docker-Kubernetes-Readiness-Liveness-Probes.php)

Docker & Kubernetes : Helm chart repository with Github pages (/DevOps/Docker/Docker_Helm_Ch

We can make container alive for a while by overwriting the CMD ["/bin/bash"], for example, sleep 30s when we run docker:

```
$ docker run ubuntu:18.04 sleep 30s

$ docker ps
CONTAINER ID  IMAGE          COMMAND       CREATED         STATUS        PORTS  NAMES
55ab52fa884d  ubuntu:18.04   "sleep 30s"   7 seconds ago   Up 6 seconds         relaxed_euler
```

But how we can make the container always run the `sleep` command when it starts? Note that we added it to the `docker run` command.

One way to avoid adding the "sleep 30s" after the command is to use the `CMD` instruction in our Dockerfile:

```
FROM ubuntu:18.04
CMD sleep 30
```

Or we can use array:

```
FROM ubuntu:18.04
CMD ["sleep", "30"]
```

Note that we should NOT use the following because the command and args should be separated:

```
CMD ["sleep 30"] X
```

Now we can build our image with a name of "ubuntu-sleep":

```
$ docker build -t ubuntu-sleep .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM ubuntu:18.04
 ---> 47b19964fb50
Step 2/2 : CMD ["sleep", "30"]
 ---> Running in c84ecc7a5b3d
Removing intermediate container c84ecc7a5b3d
 ---> 3f21ee94c150
Successfully built 3f21ee94c150
Successfully tagged ubuntu-sleep:latest
```

Then, run a container from the newly created image:

```
$ docker run ubuntu-sleep
```

The container always sleeps 30s after it started!

But we have a problem. What if we want to change the sleep time?

Currently, it's been hard-coded.

Docker & Kubernetes : Deploying WordPress and MariaDB with Ingress to Minikube using Helm Chart (/DevOps/Docker/Docker_Helm_Ch

Docker & Kubernetes : Deploying WordPress and MariaDB to AWS using Helm 2 Chart (/DevOps/Docker/Docker_Helm_Ch

Docker & Kubernetes : Deploying WordPress and MariaDB to AWS using Helm 3 Chart (/DevOps/Docker/Docker_Helm3_C

Docker & Kubernetes : Helm Chart for Node/Express and MySQL with Ingress (/DevOps/Docker/Docker_Helm_Ch

Docker & Kubernetes : Docker_Helm_Chart_Node_Expess (/DevOps/Docker/Docker_Helm_Ch

Docker & Kubernetes: Deploy Prometheus and Grafana using Helm and Prometheus Operator - Monitoring Kubernetes node resources out of the box (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Istio (service mesh) sidecar proxy on GCP Kubernetes (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Istio on EKS (/DevOps/Docker/Docker-Kubernetes-EKS-with-ISTIO.php)

Docker & Kubernetes : Deploying .NET Core app to Kubernetes Engine and configuring its traffic managed by Istio (Part I) (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Deploying .NET Core app to Kubernetes Engine and configuring its traffic managed by Istio (Part II - Prometheus, Grafana, pin a service, split traffic, and inject faults) (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Helm Package Manager with MySQL on GCP Kubernetes Engine (/DevOps/Docker/Docker_Helm_Pa

Docker & Kubernetes : Deploying Memcached on Kubernetes

Of course, we can overwrite the command like this:

```
$ docker run ubuntu-sleep sleep 5
```

However, because the image name itself is already indicating it would sleep, we need to find a way of just feeding the seconds as an argument not with the `sleep` command, and the image automatically invoke the "sleep" command needing only the parameter. Something like this:

```
$ docker run ubuntu-sleep 5
```

That's why we need the `ENTRYPOINT` instruction.

It simply specifies a program to run when a container starts.

So, our Dockerfile should be changed from:

```
FROM ubuntu:18.04
CMD ["sleep", "30"]
```

to:

```
FROM ubuntu:18.04
ENTRYPOINT ["sleep"]
```

Build a new image and run the container:

```
$ docker build -t ubuntu-sleep .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM ubuntu:18.04
 ---> 47b19964fb50
Step 2/2 : ENTRYPOINT ["sleep"]
 ---> Running in e5e6e83e9e01
Removing intermediate container e5e6e83e9e01
 ---> affbc2e6ed86
Successfully built affbc2e6ed86
Successfully tagged ubuntu-sleep:latest

$ docker run ubuntu-sleep 5
```

Note the difference between the `CMD` and `ENTRYPOINT` with related to the supplied to the `docker run` command. While the `CMD` will be completely over-written by the supplied command (or args), for the `ENTRYPOINT`, the supplied command will be appended to it.

Another problem in our Dockerfile: let's see:

```
$ docker run ubuntu-sleep
sleep: missing operand
Try 'sleep --help' for more information.
```

Engine (/DevOps/Docker/Docker_Helm_Pa

Docker & Kubernetes : EKS Control Plane (API server) Metrics with Prometheus (/DevOps/Docker/Docker-Kubernetes-EKS-Control-Plane-API-Server-Metrics-with-Prometheus.php)

Docker & Kubernetes : Spinnaker on EKS with Halyard (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Continuous Delivery Pipelines with Spinnaker and Kubernetes Engine (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes: Multi-node Local Kubernetes cluster - Kubeadm-dind(docker-in-docker) (/DevOps/Docker/Docker-Kubernetes-Multi-Node-Local-Clusters-dind.php)

Docker & Kubernetes: Multi-node Local Kubernetes cluster - Kubeadm-kind(k8s-in-docker) (/DevOps/Docker/Docker-Kubernetes-Multi-Node-Local-Clusters-kind.php)

Docker & Kubernetes : nodeSelector, nodeAffinity, taints/tolerations, pod affinity and anti-affinity - Assigning Pods to Nodes (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : Jenkins-X on EKS (/DevOps/Docker/Docker_Kuberne X-EKS.php)

Docker & Kubernetes : ArgoCD App of Apps with Heml on Kubernetes (/DevOps/Docker/Docker_Kuberne

Docker & Kubernetes : ArgoCD on Kubernetes cluster (/DevOps/Docker/Docker_Kuberne

In the command above, we did not supply an arg for the `sleep` command, and got an error when the container started.

We need a default value for the command so that container runs event though an arg is missing.

Here is where the `CMD` comes into play: the `CMD` instruction will be appended to the `ENTRYPOINT` instruction.

Here is our new Dockerfile:

```
FROM ubuntu:18.04
ENTRYPOINT ["sleep"]
CMD ["5"]
```

Build the image and run a container from the image, and we should not get any error when we do not specify sleep time:

```
$ docker build -t ubuntu-sleep .
$ docker run ubuntu-sleep
```

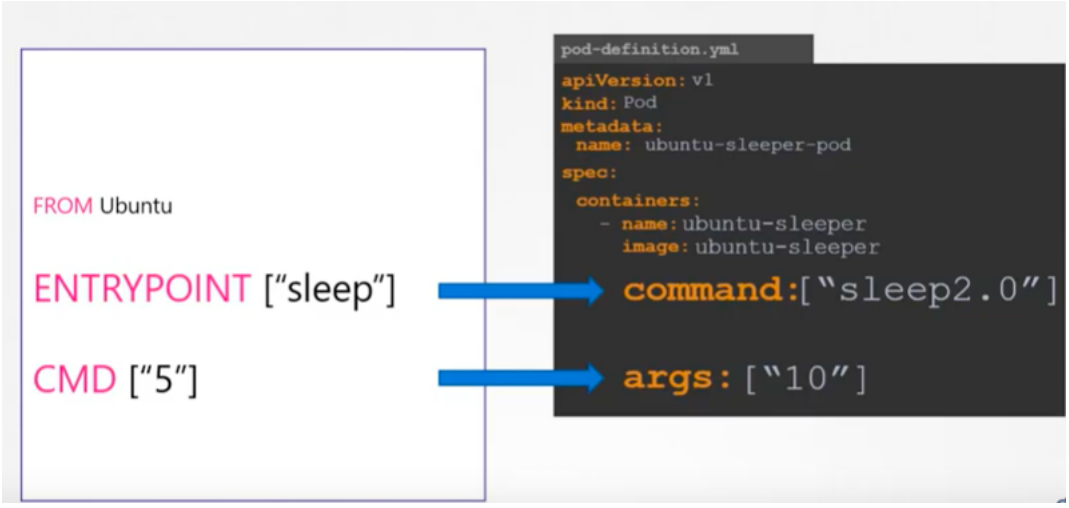
If we add a parameter to the command, it will overwrites the default value specified in `CMD`.

One more thing regarding the `ENTRYPOINT`. What if we want to override the command specified in the `ENTRYPOINT`?

In that case, we can give a new command in `docker run` command, for example:

```
$ docker run --entrypoint new-sleep-command ubuntu-sleep 60
```

Let's go further and look into how the `ENTRYPOINT` and `CMD` in Dockerfile are translated in a Pod definition yaml file:



Picture source Docker for Beginners - Commands vs Entrypoint - Kubernetes (https://www.youtube.com/watch?v=OYbEWUbmk90&index=7&list=PL2We04F3Y_43dAehLMT5GxJhtk3mJtkl5)

As we can see the parameters in `ENTRYPOINT` and `CMD` can be overwritten with the ones provided via "command" are "args" in "spec.containers" of the yaml.

Ansible 2.0

What is Ansible? (/DevOps/Ansible/Ansible_What_is

Quick Preview - Setting up web servers with Nginx, configure environments, and deploy an App (/DevOps/Ansible/Ansible_SettingU

SSH connection & running commands (/DevOps/Ansible/Ansible-SSH-Connection-Setup-Run-Command.php)

Ansible: Playbook for Tomcat 9 on Ubuntu 18.04 systemd with AWS (/DevOps/Ansible/Ansible-Tomcat9-Ubuntu18-