***DOCKER***

Uninstall old versions

$ sudo apt-get remove docker docker-engine docker.io

## Installing Docker

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt-get update

apt-cache policy docker-ce

sudo apt-get install -y docker-ce

sudo systemctl status docker

Output

docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Sun 2016-05-01 06:53:52 CDT; 1 weeks 3 days ago

Docs: https://docs.docker.com

Main PID: 749 (docker.

If you want to avoid typing sudo whenever you run the docker command, add your username to the docker group:

sudo usermod -aG docker ${USER}

To apply the new group membership, you can log out of the server and back in, or you can type the following:

su - ${USER}

You will be prompted to enter your user's password to continue. Afterwards, you can confirm that your user is now added to the docker group by typing:

id –nG

sammy sudo docker

If you need to add a user to the docker group that you're not logged in as, declare that username explicitly using:

sudo usermod -aG docker username

Using the Docker Command

docker [option] [command] [arguments]

docker –v [ for version checking ]

docker [ To view all available subcommands ]

docker docker-subcommand –-help

docker info

[Will display all info about docker images count, container count, running and stopped container count server info, logs …etc]

docker search ubuntu [search and display all images available in hub]

docker pull hello-world [ download image from docker hub ]

docker run hello-world

[Starts a container hello-world, if image not found on local machine then it will download image from hub and start the container]

docker images [ displays all downloaded images ]

docker images –q [ prints all images ID’s ]

docker rmi hello-world [ Removes image from local machine ]

docker history hello-world [ shows the layers of the images ]

docker history ubuntu [ shows layers in ubuntu image ]

docker commit -m "What did you do to the image" -a "Author Name" container-id repository/new\_image\_name

docker commit -m "added node.js" -a "Sunday Ogwu-Chinuwa" d9b100f2f636 finid/ubuntu-nodejs

docker login -u docker-registry-username

docker push docker-registry-username/docker-image-name

docker start container\_id

docker stop container\_id

docker restart container\_id

docker start $(docker ps –a –q) 🡪🡪 start all stopped containers

docker rm -f $(docker ps –a –q) 🡪🡪 remove all containers

# Docker Explained: Using Dockerfiles to Automate Building of Images

**Docker** containers are created by using base images. An image can be basic, with nothing but the operating-system fundamentals, or it can consist of a sophisticated pre-built application stack ready for launch.

When building your images with Docker, each action taken (i.e. a command executed such as apt-get install) forms a new layer on top of the previous one. These base images then can be used to create new containers.

In this DigitalOcean article, we will see about automating this process as much as possible, as well as demonstrate the best practices and methods to make most of Docker and containers via Dockerfiles: scripts to build containers, step-by-step, layer-by-layer, automatically from a base image.

DOCKER file Commands

ADD

CMD

ENTRYPOINT

ENV

EXPOSE

FROM

MAINTAINER

RUN

USER

VOLUME

WORKDIR

## Dockerfiles

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.

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.

Dockerfiles use simple, clean, and clear syntax which makes them strikingly easy to create and use. They are designed to be self explanatory, especially because they allow commenting just like a good and properly written application source-code.

**A Simple Example:**

# Print "Hello docker!"

RUN echo "Hello docker!"

## Dockerfile Commands (Instructions)

Currently there are about a dozen different set of commands which Dockerfiles can contain to have Docker build an image. In this section, we will go over all of them, individually, before working on a Dockerfile example.

### ADD

The ADD command gets two arguments: a source and a destination. It basically copies the files from the source on the host into the container's own filesystem at the set destination. If, however, the source is a URL (e.g. <http://github.com/user/file/>), then the contents of the URL are downloaded and placed at the destination.

Example:

# Usage: ADD [source directory or URL] [destination directory]

ADD /my\_app\_folder /my\_app\_folder

### CMD

The command CMD, similarly to RUN, can be used for executing a specific command. However, unlike RUN it is not executed during build, but when a container is instantiated using the image being built. Therefore, it should be considered as an initial, default command that gets executed (i.e. run) with the creation of containers based on the image.

**To clarify:** an example for CMD would be running an application upon creation of a container which is already installed using RUN (e.g. RUN apt-get install …) inside the image. This default application execution command that is set with CMD becomes the default and replaces any command which is passed during the creation.

Example:

# Usage 1: CMD application "argument", "argument", ..

CMD "echo" "Hello docker!"

### ENTRYPOINT

ENTRYPOINT argument sets the concrete default application that is used every time a container is created using the image. For example, if you have installed a specific application inside an image and you will use this image to only run that application, you can state it with ENTRYPOINT and whenever a container is created from that image, your application will be the target.

If you couple ENTRYPOINT with CMD, you can remove "application" from CMD and just leave "arguments" which will be passed to the ENTRYPOINT.

Example:

# Usage: ENTRYPOINT application "argument", "argument", ..

# Remember: arguments are optional. They can be provided by CMD

# or during the creation of a container.

ENTRYPOINT echo

# Usage example with CMD:

# Arguments set with CMD can be overridden during \*run\*

CMD "Hello docker!"

ENTRYPOINT echo

### ENV

The ENV command is used to set the environment variables (one or more). These variables consist of “key value” pairs which can be accessed within the container by scripts and applications alike. This functionality of Docker offers an enormous amount of flexibility for running programs.

Example:

# Usage: ENV key value

ENV SERVER\_WORKS 4

### EXPOSE

The EXPOSE command is used to associate a specified port to enable networking between the running process inside the container and the outside world (i.e. the host).

Example:

# Usage: EXPOSE [port]

EXPOSE 8080

To learn about Docker networking, check out the [Docker container networking documentation](https://docs.docker.com/engine/userguide/networking/).

### FROM

FROM directive is probably the most crucial amongst all others for Dockerfiles. It defines the base image to use to start the build process. It can be any image, including the ones you have created previously. If a FROM image is not found on the host, Docker will try to find it (and download) from the **Docker Hub** or other container repository. It needs to be the first command declared inside a Dockerfile.

Example:

# Usage: FROM [image name]

FROM ubuntu

### MAINTAINER

One of the commands that can be set anywhere in the file - although it would be better if it was declared on top - is MAINTAINER. This non-executing command declares the author, hence setting the author field of the images. It should come nonetheless after FROM.

Example:

# Usage: MAINTAINER [name]

MAINTAINER authors\_name

### RUN

The RUN command is the central executing directive for Dockerfiles. It takes a command as its argument and runs it to form the image. Unlike CMD, it actually **is** used to build the image (forming another layer on top of the previous one which is committed).

Example:

# Usage: RUN [command]

RUN aptitude install -y riak

### USER

The USER directive is used to set the UID (or username) which is to run the container based on the image being built.

Example:

# Usage: USER [UID]

USER 751

### VOLUME

The VOLUME command is used to enable access from your container to a directory on the host machine (i.e. mounting it).

Example:

# Usage: VOLUME ["/dir\_1", "/dir\_2" ..]

VOLUME ["/my\_files"]

### WORKDIR

The WORKDIR directive is used to set where the command defined with CMD is to be executed.

Example:

# Usage: WORKDIR /path

WORKDIR ~/

## How to Use Dockerfiles

Using Dockerfiles is as simple as having the Docker daemon run one. The output after executing the script will be the ID of the new docker image.

Usage:

# Build an image using the Dockerfile at current location

# Example: docker build -t [name] .

docker build -t my\_mongodb .

## Dockerfile Example: Creating an Image to Install MongoDB

## In this final section for Dockerfiles, we will create a Dockerfile document and populate it step-by-step with the end result of having a Dockerfile, which can be used to create a docker image to run MongoDB containers.

**Note:** After starting to edit the Dockerfile, all the content and arguments from the sections below are to be written (appended) inside of it successively, following our example and explanations from the **Docker Syntax** section. You can see what the end result will look like at the latest section of this walkthrough.

############################################################

# Dockerfile to build MongoDB container images

# Based on Ubuntu

############################################################

### Setting The Base Image to Use

# Set the base image to Ubuntu

FROM ubuntu

### Defining The Maintainer (Author)

# File Author / Maintainer

MAINTAINER Example McAuthor

### Setting Arguments and Commands for Downloading MongoDB

################## BEGIN INSTALLATION ######################

# Install MongoDB Following the Instructions at MongoDB Docs

# Ref: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/

# Add the package verification key

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

# Add MongoDB to the repository sources list

RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | tee /etc/apt/sources.list.d/mongodb.list

# Update the repository sources list

RUN apt-get update

# Install MongoDB package (.deb)

RUN apt-get install -y mongodb-10gen

# Create the default data directory

RUN mkdir -p /data/db

##################### INSTALLATION END #####################

### Setting The Default Port For MongoDB

# Expose the default port

EXPOSE 27017

# Default port to execute the entrypoint (MongoDB)

CMD ["--port 27017"]

# Set default container command

ENTRYPOINT usr/bin/mongod

### Saving The Dockerfile

After you have appended everything to the file, it is time to save and exit. Press CTRL+X and then Y to confirm and save the Dockerfile.

This is what the final file should look like:

############################################################

# Dockerfile to build MongoDB container images

# Based on Ubuntu

############################################################

# Set the base image to Ubuntu

FROM ubuntu

# File Author / Maintainer

MAINTAINER Example McAuthor

################## BEGIN INSTALLATION ######################

# Install MongoDB Following the Instructions at MongoDB Docs

# Ref: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/

# Add the package verification key

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

# Add MongoDB to the repository sources list

RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | tee /etc/apt/sources.list.d/mongodb.list

# Update the repository sources list

RUN apt-get update

# Install MongoDB package (.deb)

RUN apt-get install -y mongodb-10gen

# Create the default data directory

RUN mkdir -p /data/db

##################### INSTALLATION END #####################

# Expose the default port

EXPOSE 27017

# Default port to execute the entrypoint (MongoDB)

CMD ["--port 27017"]

# Set default container command

ENTRYPOINT usr/bin/mongod

### Building Our First Image

Using the explanations from before, we are ready to create our first MongoDB image with docker!

docker build -t my\_mongodb .

**Note:** The **-t [name]** flag here is used to tag the image. To learn more about what else you can do during build, run docker build --help.

### Running A MongoDB Instance

Using the image we have build, we can now proceed to the final step: creating a container running a MongoDB instance inside, using a name of our choice (if desired with **-name [name]**).

docker run -name my\_first\_mdb\_instance -i -t my\_mongodb