Running dockter-tom & LAMP on Debian Stretch

29 March 2018 Version 1.0

1. First, uninstall older versions of Docker called “docker” or “docker-engine“ along with associated dependencies. If your system does not have a Docker package, skip the step below.
   1. # apt-get -y remove docker docker-engine docker.io
2. Next, install docker CE (Community Edition) from docker.com. Directions follow.
3. Install Prerequisites: Note: Docker needs a 64-bit version of Debian OS and Kernel version should be at least 3.10.

# apt-get install apt-transport-https dirmngr ca-certificates wget software-properties-common

1. You are going to add the docker repo to your /etc/apt/sources.list sources list & then get GPG key:

# wget <https://download.docker.com/linux/debian/gpg>

# apt-key add gpg

1. Obtain Docker’s repository signature and updated package index (NOTE: The backslashes in this command indicate non-breaking space. The command below should all be on one line):

# echo "deb [arch=amd64] <https://download.docker.com/linux/debian> \ $(lsb\_release -cs) stable" | sudo tee –a \ /etc/apt/sources.list.d/docker.list

# apt-get update

Make sure you are installing Docker from the official repository, not from the default Debian repository.

# apt-cache policy docker-ce

1. You should see the output like below with the Docker repository details.

docker-ce:

Installed: (none)

Candidate: 18.03.0~ce-0~debian

Version table:

17.06.0~ce-0~debian 500

500 https://download.docker.com/linux/debian stretch/stable amd64 Packages

17.03.2~ce-0~debian-stretch 500

500 https://download.docker.com/linux/debian stretch/stable amd64 Packages

17.03.1~ce-0~debian-stretch 500

500 https://download.docker.com/linux/debian stretch/stable amd64 Packages

17.03.0~ce-0~debian-stretch 500

500 https://download.docker.com/linux/debian stretch/stable amd64 Packages

1. Install Docker

# apt-get install docker-ce

1. Controlling the Docker service

* To start Docker, run:
  + systemctl start docker
* To stop Docker service, run:
  + systemctl stop docker
* To restart Docker service, run:
  + systemctl restart docker
* To check the status of Docker service, run:
  + systemctl status docker
* To enable Docker service to autostart on system boot, run:
  + systemctl enable docker

1. To verify your Docker Installation you will run a “hello-world” container. Run:

docker run hello-world

The “hello-world” image is likely not on your computer, so it will be pulled automatically from Docker Hub (<https://hub.docker.com/> ).

The output below confirms that you have correctly installed Docker on Debian OS.

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

b04784fba78d: Pull complete

Digest: sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

1. Now you are ready to run dockter-tom or LAMP. To start LAMP, run:
   1. docker run -dit -p 22:22 -p 80:80 -p 443:443 -p 3306:3306 lamp
2. It will take 10-20 minutes to build depending upon the speed of your Internet connection. Note: If you are running sshd, httpd, or mysql/mariaDB on your host machine, you will need to disable these services before spinning up the LAMP container. Otherwise, the container will fail to build correctly because we are mapping its internal ports to the external ports: 22, 80, 443, & 3306
3. After the build completes, you can inspect your container. Run:
   1. docker ps (this will give you the container ID, which looks something like this: “**30cefef74ba6”**
   2. docker inspect <container\_id>
4. If you want to prettify the output, install “jq” by running: apt-get install jq –y . Don’t include the period in the previous command. You can then pipe the JSON output of docker inspect to jq. Run:
   1. docker inspect <container\_id> | jq
5. You can also test the LAMP container by running the command:
   1. docker exec –it <container\_id) /bin/bash
6. This command will enable you to use the BASH shell inside of the container. You can observe this by noting that your prompt changes to: <container\_id:/#
7. To check if sshd, http, https, & mysql are running from inside your container, you can use the following commands:
   1. ps waux | grep ssh
   2. ps waux | grep apache
   3. ps waux | grep mysql
8. You should also test the container, by typing exit to drop you back into the host shell. Then run a browser agaist <http://localhost> & <https://localhost>. You can check that mysql is installed & configured by clicking through the Wordpress install, & using wp\_datebase for the Database name; wp\_user for the Username; & wp\_password for the Password field. This will dump you into the famous WordPress 5 minute install screen.
9. A container can exist in a “stopped” state & will not show up if you use the command: docker ps
10. You can check for containers in all states by running: docker ps –aq
11. Below is a short shell script that you can use if you want to clean up your system after installing the LAMP container. You can then rebuild in a clean environment.

BEGIN ----------------

#!/bin/bash

set -eo pipefail

## rmdock.sh

## Simple utility script to clean up test environment

## No error checking.

docker stop $(docker ps -a -q)

docker rm $(docker ps -a -q)

docker system prune --volumes -f

echo $(docker system df)

exit 1

END -------------

Below are some docker commands that can be useful. They are in cheat sheet form.

By default docker containers run as root

AWS: service docker start

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

If you just want to make some modification to files or inspect processes, here's one another solution you probably want.

You could run the following command to execute a new process from the existing container:

docker exec -ti [CONTAINER-ID] bash

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

##### LIST DISK SPACE USED BY ALL DOCKER COMPONENTS

docker system df

echo $(docker system df)

#################### CONTAINERS ###########################

##### ASSIGN A NAME TO A CONTAINER ########

docker run -d --name <CONTAINER ID> nameA/nameB

##### LIST ALL EXITED CONTAINERS

docker ps -aq -f status=exited

##### LIST ACTIVE CONTAINERS

docker ps -aq

##### STOP ALL DOCKER CONTAINERS

docker stop $(docker ps -a -q)

##### STOP A SINGLE DOCKER CONTAINER

docker stop <container ID>

##### REMOVE ALL CONTAINERS -- AFTER STOPPING ALL

docker rm $(docker ps -a -q)

docker rm $(docker ps -qa --no-trunc --filter "status=exited")

##### REMOVE A SINGLE CONTAINER

docker rm <container ID>

##### RESTART ONE OR MORE CONTAINERS

docker restart <CONTAINER\_ID>

##### PURGE ALL UNUSED IMAGES, CONTAINER, VOLUMES, & NETWORKS

docker system prune

################ IMAGES ###################

##### LIST DOCKER IMAGES

docker images -a

docker images -a | grep "PATTERN"

##### REMOVE DOCKER IMAGES

docker rmi <IMAGE>

docker images -a | grep "PATTERN" | awk '{print $3}' | xargs docker rmi

docker rmi $(docker images --filter "dangling=true" -q --no-trunc)

docker rmi $(docker images --filter "dangling=true" -q --no-trunc) 2>/dev/null

##### LIST DANGLING IMAGES

docker images -f dangling=true

##### REMOVE DANGLING IMAGES

docker images purge

##### PURGE ALL UNUSED IMAGES, CONTAINER, VOLUMES, & NETWORKS

docker system prune

####################### VOLUMES #######################

##### LIST VOLUMES

docker volume ls

##### LIST DANGLING VOLUMES

docker volume ls -f dangling=true

##### REMOVE VOLUMES

docker volume rm volume\_name volume\_n

docker volume rm `docker volume ls -q -f dangling=true`

############### DOCKER NETWORKING ##################

##### LIST DOCKER NETWORKS

docker network ls

##### INSPECT DOCKER NETWORKING

docker network inspect -f '{{range .Containers}}{{println .Name .IPv4Address}}{{end}}' <NETWORK NAME>

##### RETURN INFO ABOUT DOCKER NETWORK

docker network inspect bridge

##### DELETE ALL DOCKER NETWORKS

docker network rm $(docker network ls | grep "bridge" | awk '/ / { print $1 }')

################ CLEAN SLATE ##############

##### DELETE EVERY DOCKER CONTAINER

##### Must be run first because images are attached to containers

docker rm -f $(docker ps -a -q)

##### DELETE EVERY DOCKER IMAGE

docker rmi -f $(docker images -q)

docker system prune -- FOR GOOD MEASURE

##### DELETE ALL DOCKER NETWORKS

docker network rm $(docker network ls | grep "bridge" | awk '/ / { print $1 }')

################# DOCKER SYSTEM COMMANDS ###################

##### PULL AN IMAGE FROM DOCKER.IO

docker pull <IMAGE-NAME>

##### PULL AN IMAGE FROM A DIFFERENT REPOSITORY

docker pull <HOSTNAME ie reg.domain.com>/<IMAGE-NAME>

##### DOCKER EXEC - RUN COMMANDS WITHIN A CONTAINER

docker exec -it <container\_id\_or\_name> echo "Hello from container!"

##### START A NEW BASH SHELL IN CONTAINER

docker exec -it <container\_id\_or\_name> bash

##### SET AN ENV VARIABLE IN CONTAINER

docker exec -it -e VAR=1 <container\_id\_or\_name> bash

##### COPY A FILE FROM CONTAINER TO HOST

docker cp <CONTAINER ID>:/path/to/container/file host/path/file

##### DOCKER LOG COMMANDS

docker log -f <CONTAINER ID>

##### DOCKER INSPECT - WORKS IF CONTAINER IS STOPPED OR EXITED

docker inspect <CONTAINER ID> | jq

################## MOVING DOCKER CONTAINERS ###################

NOTE: These docker export does not export the contents of volumes associated with the container

##### EXPORT A DOCKER CONTAINER (RUNNING OR PAUSED)

docker export <CONTAINER ID> -o CONTAINER-NAME.tar

##### EXPORT A DOCKER image

docker save -o IMAGE-NAME.tar <IMAGE-NAME>

##### IMPORT A DOCKER CONTAINER (OPPOSITE OF EXPORT)

docker import CONTAINER-NAME.tar

##### IMPORT A DOCKER IMAGE (OPPOSITE OF SAVE)

docker load --input IMAGE-NAME.tar

##### RUN DOCKER WITHOUT SUDO #############

When you install docker, you have to use sudo to work with docker commands. To avoid using sudo, you need to add your current user to docker group using the following command. Restart the terminal for the changes to take place.

usermod -aG docker ${USER}

usermod -aG docker ${USER}

Execute the following command if you do not want to restart the terminal:

newgrp docker

newgrp docker

Of course, you can also use, "sudo su"

########### ENABLE DOCKER REST API FOR NET & SOCKETS #########

Open the file/lib/systemd/system/docker.service, search for ExecStart and add values below.

ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4444 -H unix:///var/run/docker.sock

sudo systemctl daemon-reload

sudo service docker restart

######################### DOCKER STATS #######################

curl --unix-socket /var/run/docker.sock http:/containers/json | python -m json.tool

STREAMING STATS: curl --unix-socket /var/run/docker.sock http:/containers/8a9973a456b3/stats

ONE SHOT: curl --unix-socket /var/run/docker.sock http:/containers/8a9973a456b3/stats?stream=false

cAdvisor is a simple server that taps the Docker API and provides one minute of historical data in 1-second increments. It’s a useful way to visualize what’s going on at a high level with your Docker containers on a given host. cAdvisor simply requires one container per host that you’d like to visualize.

sudo docker run \

 --volume=/:/rootfs:ro \

 --volume=/var/run:/var/run:rw \

 --volume=/sys:/sys:ro \

 --volume=/var/lib/docker/:/var/lib/docker:ro \

 --publish=8080:8080 \

 --detach=true \

 --name=cadvisor \

 google/cadvisor:latest

################# SHARING A VOLUME #################

Sharing Volumes Between Containers

It is possible to share the volumes created by one container with another by using the --volumes-from parameter to docker run. For example, say we make a container named 'ContainerA’ that has a volume:

# docker run -i -v /var/volume1 -name 'ContainerA' -t fedora /bin/bash

We can share the volumes from this container with another container:

# docker run -i --volumes-from ContainerA -t fedora /bin/bash

In ContainerB, we will see /var/volume1 from ContainerA. For more information about sharing volumes, please check out the docker documentation

When to Use Volumes

We recommend that you use volumes in the following use-cases:

You want to be able to share a directory between containers

You intend on writing large amounts of data to a directory, for example, for a database