**<https://github.com/docker/labs/tree/master/beginner>**

[**https://collaborate.citi.net/groups/containers-docker-at-citi**](https://collaborate.citi.net/groups/containers-docker-at-citi)

**Ubuntu:**

1. sudo su
2. service docker.io status
3. apt-get update
4. apt-get install –y docker.io
5. service docker.io status
6. docker –v
7. docker version
8. docker info
9. wget –qO- <https://get.docker.com/gpg> | apt-key add -
10. echo deb <http://get.docker.com/ubuntu> docker main > /etc/apt/sources.list.d/docker.list
11. apt-get update
12. apt-get install lxc-docker
13. docker –v
14. docker run –it ubuntu /bin/bash
15. sudo gpasswd –a jithu docker (logoff and login back)
16. docker run –it ubuntu /bin/bash
17. exit

**CentOS:**

1. yum install –y docker
2. systemctl status docker.service
3. systemctl start docker.service
4. systemctl status docker.service

Communication over N/W

Ubuntu:

1. netstat –tlp (to check if a docker program is listening on a port)
2. service docker stop
3. docker –H <IP>:2375 –d &
4. netstat –tlp
5. docker info (it will fail)
6. After testing below (1 to 3) : docker –H <IP>:2375 –H unix:///var/run/docker.sock –d & (to listen network port as well as the local unix socket)

CentOS:

1. export DOCKER\_HOST=”tcp://<ip>:2375”
2. docker version (server version will be the one from Ubuntu)
3. export DOCKER\_HOST=

Playing around with some container:

Ubuntu:

1. docker run –it centos /bin/bash

**Images:**

* images are we what we load containers from
* docker run –it fedora /bin/bash
* docker pull fedora
* docker pull –a fedora
* docker images fedora
* docker pull coreos/etcd
* Once downloaded, images are stored under /var/lib/docker/>storage drive> in this case aufs. (Ubuntu)
* docker images
* images are design time, container and run time. In other words, containers are the run time instances of an image.

**Containers:**

* Launched from images, running instances of an image.
* docker run –it defora /bin/bash (it will create the container)
* docker attach <container#>
* Ctrl + P +Q (for existing from a container)
* docker ps
* docker ps –a (history as well)

**Registries and Repos:**

* Images are stored in repo, and repo is under registry,
* Default registry is hub.docker.com
* Registry.hub.docker.com (in browser) - dicker Inc and Canonical maintaining it.

**Layering**

* Union mounts
* Multiple Layers.
* Top Layer is the only R/W layer (Copy and write)
* Top layer wins any conflict.
* Rootfs on bottom (some time bootfs under that)
* docker images –tree
* You can see these under /var/lib/docker/aufs/layers/. Cat on the later, to see the layers beneath it.
* The actual files can be seen udner /var/lib/docker/aufs/diff/<layer#>

*Copying container from one host to another*

* From Ubuntu to CentOs, usinf old fashion ( creating tar and copying it.)
* docker images
* docker run Ubuntu /bin/bash –c “echo ‘cool content’ > /tmp/coolFile”
  + No “-i” cus short lived command. We don’t need an interactive shell.
  + Detached mode.
* docker ps –a
* docker commit <ID> fridge
  + A new image with the name fridge has been created.
* docker images
  + You can see fidge
* docker images –-tree
* docker history fridge
* docker save –o /tmp/fridge.tar fridge

*Copy that in centOS*

* tar –tf /tmp/fridge.tar
  + Each image has a directory, which contains a metadata and a tar ball.
* docker load –i /tmp/fridge.tar
* docker images
* docker run –it fridge /bin/bash
* check the file /tmp/cool-file

**More on container**

* When we say”run”, the docker engines creates a container from the image, by loading the different layers based on the order mentioned on the metadata.
* Each container will have its own thin writable top layer, over all the image layer.
* rootfs is read-only, however because of the feature of the union mount, the top writable layer gives the look and feel of a rootfs.
* docker run –d Ubuntu /bin/bash –c “ping 8.8.8.8 –c 30” (-d is for detached mode)
* docker ps
* docker top <ID/Name of the container>
* One process per container (recommended), we can run many though.
* container exist while the process running in the container exist. As soon as that process exits, the container will exit.
* docker run Ubuntu:14.04 (specifying the exact version)
* docker run –-cpu-shares=256
* docker run memory=1g
* docker run –d Ubuntu:14.04.1 /bin/bash –c “ping 8.8.8.8”
* docker ps
* docker inspect <ID>
* docker attach <name/ID>

**Container Management**

* docker run –it ubuntu:14.04 /bin/bash
* Ctrl + P + Q
* docker ps
* docker stop <ID/name> (sends a SIGTERM)
* docker ps
* docker kill –s <SIGNAL> (send a SIGKILL)
* will go to the container, to the process running with PID1.
* docker ps –a
* docker ps –l
* docker start <ID/name.
* docker attach <ID/name> (Attaches to PID1 of the container)
* ctrl+P+Q
* docker restart <ID/name>
* docker ps
* docker info
* docker rm <id/name>
* you cannot remove a running container.
* docker rm –f <id>

**Inside of a container**

* docker top <id/name>
* docker attach <id/name>
  + and ten do “ps –ef”
  + ctrl+P+Q
* docker log <id/name>
* docker inspect <id/name>
* two JSON files.
* ls –l /var/lib/docker/containers/<fullname> ; you will see config.jspn and hostconfig.json
* docker attach : attaches to PID1 in the container. Not necessarily a shell.
* nsenter
  + Allows to enter Namespace
  + Requires the container’s PID (not the ID or name)
  + We can get from docker inspect.
  + docker inspect <id/name> | grep pid
  + nsenter –m –u –n –p –I –t <pid> /bin/bash ( mount ns, utc ns, n/w ns, Process nsp, ipc ns, target)
  + you can now enter the container, and if you do exit, you will only exit from sheel, and container will continue to run.
  + docker-enter <id/name>
  + exit
  + docker exec it <id/name> /bin/bash (recommended)
  + exit

if “nsenter” is not installed;

* + docker run –v /usr/local/bin:/target jpetazzo/nsenter

**Docker Images from Dockerfiles**

* Dockerfile
* Plain-text
* Simple format
* Instructions to build image
* Run “docker build”
* <https://youtube.com/watch?v=wW9CAH9nSLs>
* FROM should be the first instruction.
* MAINTAINER – email ID of the maintainer. Can be anywhere in the docker file. Better to use instruction in UPPER case.
* RUN – for running commands against the images that we are building; every Run commands create a layer.
* CMD – commands.
* Any files or directories present in the directory where Dockerfile exists, will automatically get included in the image.
* From the directory where the Dockerfile exists
  + docker build –t helloworld:0.1 . ( “.” For PWD, and no upper case for the image name)
  + docker images –tree
    - you will see the new image under the Ubuntu Image
  + docker history <imageID>
  + docker run <imageName> : create the container

**Working with Registry**

* current stable version is written in Python
* that will be superseded by a version written in Go.

*Pushing an image to the registry*

* Create a repo (from the hub)
* docker images
* docker tag <imageId> jithus/helloworld:1.0
* docker build –t (to tag during the build)

*public registry:*

* docker push jithus/helloworld:1.0 (Only new layers get pushed)
* To delete an image, the containers built using the images have to be removed first.
* docker rm <list of all container IDs separated by space>
* docker rmi <list of image IDs>
* docker pull jithus/helloworld:1.0
* registry v1 :python (current stable)
* regtistry v2: golang, currently under development (Docker itself is written in Golang)

*private registry:*

* in Debian
  + docker run –d –p 5000:5000 registry (-d detached, -p port 5000 inside the container is exposed , and make it available on port 5000 on docker host)
* In Ubuntu
  + docker images
  + docker ps
  + <hostname>.docker.course:5000 in browser.
  + (side note: docker config file under /etc/default/docker has entry DOCKER\_OPTS which will have SSL settings; DOCKER\_OPTS=”-–insecure-registry <hostname>.docker.course:5000
  + Docker push <hostname>.docker.course:5000/<imageName>
* On centOs : docker run –d <hostname>.docker.course:5000/<imageName>
  + Side Note: /usr/lib/system/system/docker.service
  + ExecStart=/usr/bin/docker –d $OPTIONS $DOCKER\_STORAGE\_OPTIONS -–insecure-registry <hostname>.docker.course:5000
* DHE (Docker Hun Enterprise)

**Docker Images from Dockerfiles in Detail**

* BuildCache
  + Clean up the images and containers, cd /pluralsight, and you have a Dockerfile there.
  + docker images
  + docker info
  + docker images --tree
  + docker build –t=”build1” .
  + docker build –t=”build2” . (will be faster this time becase of the build cache)
  + docker info (you can see the images# increased)
  + docker images –tree (get the last image)
  + docker history <last image ID>.
* Building a web server Dockerfile
  + mkdir ~/web
  + cd ~/web
  + vi Dockerfile
  + Add CMD to run Apache on port 80
  + Build it
  + Create a container lisetning on 80 mapped to 80 of the container
  + Check browser for hostname:80
  + Combing the commands will also result in lesser number of layers as well as smaller images
  + RUN : buld time, add layers to the image, used for app install etc
  + CMD : Run time, in containers when launched, equivalent of docker run <args> <cmd>, One CMD per Dockerfile, can be overriden by command line instrctions during runtime
    - Shell form: expressed same way as shell commands, variable expansion, get prepended by “/bin/bash –c” (eg: CMD echo $var1)
    - Exec form: JSON Array style, no need of shell, no variable expansion though. [“command”,”arg1”]
  + ENTRYPOINT :
    - just like CMD, but cannot be overridden during by cmdline commands.
    - The strting at the end of the run command will be taken as an argument to the command specified in ENTRYPOINT.
    - Can be overridden by specifying “—entrypoint”
  + ENV – passing environment variables
    - ENV variable=value
  + VOLUME – decoupling of the data from containers
    - docker volume commands
    - docker run –it –v /test-vol –name=voltainer ubuntu:15.04 /bin/bash
    - created a file under /test-vol
    - docker inspect voltainer (check for Volumes and VolumesRW)
    - docker run –it –-volume-from=voltainer ubuntu:15.04 /bin/bash
    - ls –l will show the file that we created earlier.
    - To mount a file sytem on the host to the container (docker run –v /data:/data)
    - In docker file : VOLUME /data; we cannot do mount host volume from Dockerfile
    - docker rm –v conatinerName (if we don’t specify –v, the volume wont get deleted)
* Docker0 bridge
  + ip a
    - ignore lo, etho0, eth1, but you will also see a docker0.
    - Docker daemon when starts a bridge/virtual bridge called docker0, created inside the linux kernel. Passes packets beteen connected devices.
  + bridge-utils packge : apt-get install –y bridge-utils; yum install bridge-utils
    - brctl show docker0
    - docker images (net-img image)
    - docker run –it –-name=net1 net-img
      * brctl show docker0 (each new container gets one interface automatically atatched to docker0 brodge)
    - docker attach net1
    - ip a
* Network configuration files
  + hosts and resolve.conf under the /var/lib/containers/<id>
  + resolve.conf by default is a copy of the /etc/resolve.conf : Those can be overridden. every container get a copy of the resolve and hosts.
  + on the run/fly update is possible on these files, but they are no restart persistent.
  + docker run –dns=8.8.4.4 –name=dnstest net-img
* Exposing ports from a container
  + Create Dockerfile which has a “EXPOSE 80” and ENTRYPOINT “apache2ctl” and CMD “-D FOREGROUND”
  + Build : docker build –t=”apache-img” .
  + Launch a container: docker run –d –p 5001:80 --name=web1 apache-img (expose port 5001 of the host to the port 80 of the container)
* Viewing exposed ports
  + docker port web1
  + docker ps
  + tcp or udp? docker run –d –p 5002:80/udp –-name=web2 apache-img (it will show 80/ucd 🡪 0.0.0.0:5002)
  + docker run –d –p <IP>:5003:80 –-name=web2 apache-img (it will show 80/tcp 🡪 <ip>:5002)
  + If we have many ports in EXPOSE, use –P ; docker run –d **–P** –-name=web3 apache-img (it will map all the ports in the container to some HIGH numbered ports on the host)
* Linking Containers
  + Networking container, instead of exposing ports
  + Only between container, cannot be used for communicating with the outside world.
  + Source and receive containers
    - Source will have an EXPOSE 80. The container will be created withoiting mapping to any host ports : docker run –-name=src –d img
    - rcvr container will be created, by linking it to source. : docker run –-name=rcvr –-link=src:alias-src –it ubentu:15.04 /bin/bash
    - docker inspect rcvr (It will show Links: xxx)
    - docker attach rvcr
    - env | grep ALI; cat /etc/hosts

troubleshooting

* Docker daemon logging
  + Debug, info, error and fatal (Debug is the superset, and the others in the order mentioned)
  + service docker stop
  + docker –d –l debug &
  + You will see logs in the console
  + vi /etc/default/docker : DOCKER\_OPTS=”--log-level=fatal”
  + kill -9 <PID>
  + service docker start
* Container logging
  + docker log <containerName>
  + docker log –f <name> (like tail -f)
  + these logs are written in to the stdout, and docker daemon take these and add to a JSPN file. The docker log command reads this JSON file.
* Image troubleshooting
  + Run the commands outside, before executing the build; just like testing the indivual commands of a shell script instead of executing the whole script.
    - FROM ubuntu:15.04 (1)
    - RUN apt-get update (3)
    - RUN apt-get install –y iputils-ping (4)
    - CMD ping 8.8.8.8 (2)
  + You can now add the RUN commands in a single line to reduce the number of layers.
  + Now put all these in Dockerfile and build
* Intermediate Images
  + If a build fails, the “docker images” command will show some images which has the REPOSITORY and TAG as <none>. It will have an image ID which is the last succeful intermediate image. Looking at the logs and locating this image ID in the log will tell us which is the last successful command in the Dockerfile.
  + Docker run –it <last succeful image id> /bin/bash
  + Now you can start executing the further commands in the Dockerfile and figure out the issue.
* Network troubleshooting
  + Docker0 bridge down troubleshtting
  + ip a (find out the IP for docker0)
  + service docker stop
  + ip link del docker0
  + edit the /etc/default/docker : DOCKER\_OPTS=--bip=150.150.0.1/24 (Some other IP other than see above)
  + service docker start
  + ip a (you will see different IP for docker0)
  + docker run –it ubuntu:15.04 /bin/bash
  + ip a
* Firewall/IP tables troubleshooitng
  + –icc :
  + -–iptables :
  + Both defaults to true
  + iptables –L –v
  + service docker stop
  + docker file : DOCKER\_OPTS=--icc=false
  + service docker start
  + iptables -L –v (you can
  + service docker stop
  + DOCKER\_OPTS=”--icc=true -–iptables=false”
  + Service docker start
  + Spin up two containers
  + Docker inspect container1 : to get the IP address
  + Docker attach container2
  + Ping IP address of container1, and It wont work, since iptables are blocking it.