Gmail **madhavarao s <madhava567@gmail.com>**

Docker

1 message

venkat0431 kollu <kolluvenky07@gmail.com> Tue, Feb 20, 2018 at 10:44 PM
To: venkat0431 kollu <kolluvenky07@gmail.com>, madhavarao s <madhava567@gmail.com>

Hi,

https://docs.docker.com/get-started/

yum install docker
cd /var/lib/docker(location)
(systemctl start httpd.service)
(systemctl status firewalld)
https://www.youtube.com/watch?v=hwkqju_BXEo&list=PLkA60AVN3hh_nihZ1mh6cO3n-uMdF7UlV
Curl localhost:8080
cat /proc/version(check version & flavor of unix)

```
  --config=~/.docker            Location of client config files
  -D, --debug                   Enable debug mode
  -H, --host=[]                 Daemon socket(s) to connect to
  -h, --help                    Print usage
  -l, --log-level=info          Set the logging level
  --tls                         Use TLS; implied by --tlsverify
  --tlscacert=~/.docker/ca.pem  Trust certs signed only by this CA
  --tlscert=~/.docker/cert.pem  Path to TLS certificate file
  --tlskey=~/.docker/key.pem    Path to TLS key file
  --tlsverify                   Use TLS and verify the remote
  -v, --version                 Print version information and quit
```

docker images
docker ps -a
docker ps -aq(it shows container ids)
docker rm containerid
docker rmi imagename
docker pull imagename:latest
docker image pull imagename
docker run -it ubuntu bash
docker stop containerid
docker run --restart=always centos
docker rm containerid
docker run centos ls -l(see the container inside files)
docker container run ubuntu ls -l
docker container ls -a
docker container run -it ubuntu /bin/sh
docker run -i -t imagesname /bin/bash (enter to container)
docker start centos
docker kill $(docker ps -q)   (kill all container)
docker rm -v $(docker ps -a -q -f status=exited) ==>(Delete stopped containers)
docker run --rm ubuntu env (Get Environment Settings)
docker run -p 1000:80 onename (changing port no)
docker run -d -p 1000:80 onename (mapping port detached mode)
dockdocer inspect containerid (used to get metadata on a container)
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm (delete old container)
docker rmi $(docker images -q(delete all images)
docker images -viz | dot -Tpng -o docker.png (Show image dependencies)
docker exec -it [container-id] bash
docker volume ls
docker inspect $(dl) | grep IPAddress | cut -d "" -f 4
portmapping : docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}}
{{end}}' <containername>

Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run docker image inspect alpine. In the demo above, you used the docker image pull command to download the alpine image. When you executed the command docker container run hello-world, it also did a docker image pull behind the scenes to download the hello-world image.
Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using docker run which you did using the alpine image that you downloaded. A list of running containers can be seen using the docker container ls command.
Docker daemon - The background service running on the host that manages building, running and distributing Docker containers.
Docker client - The command line tool that allows the user to interact with the Docker daemon.
Docker Store - Store is, among other things, a registry of Docker images. You can think of the registry as a directory of all available Docker images. You'll be using this later in this tutorial.
docker start starts a container so it is running.
docker stop stops a running container.
docker restart stops and starts a container.
docker pause pauses a running container, "freezing" it in place.
docker unpause will unpause a running container.
docker wait blocks until running container stops.
docker kill sends a SIGKILL to a running container.
docker attach will connect to a running container.
docker cp copies files or folders between a container and the local filesystem.
docker export turns container filesystem into tarball archive stream to STDOUT.

$ docker run -d --name redis example/redis --bind 127.0.0.1
$ # use the redis container's network stack to access localhost
$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1
docker run -it --add-host db-static:86.75.30.9 ubuntu cat /etc/hosts

Give access to a single device:-

docker run -it --device=/dev/ttyUSB0 debian bash

Give access to all devices:-

docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash

Lifecycle:-

docker images =shows all images.
docker import =creates an image from a tarball.
docker build =creates image from Dockerfile.
docker commit =creates image from a container, pausing it temporarily if it is running.
docker rmi =removes an image.
docker load =loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
docker save =saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

Info:-

docker history =shows history of image.
docker tag =tags an image to a name (local or registry).

Load/Save image:-

Load an image from file:docker load < my_image.tar.gz
Save an existing image:docker save my_image:my_tag | gzip > my_image.tar.gz

Import/Export container:-

Import a container as an image from file:cat my_container.tar.gz | docker import - my_image:my_tag
Export an existing container:docker export my_container | gzip > my_container.tar.gz

network:-

docker network create
docker network rm
docker network ls
docker network inspect

docker network connect
docker network disconnect

You can specify a specific IP address for a container:

```
# create a new bridge network with your subnet and gateway for your ip block
docker network create --subnet 203.0.113.0/24 --gateway 203.0.113.254 iptastic

# run a nginx container with a specific ip in that block
$ docker run --rm -it --net iptastic --ip 203.0.113.2 nginx

# curl the ip from any other place (assuming this is a public ip block duh)
$ curl 203.0.113.2
```

Registry & Repository:-
docker login to login to a registry.
docker logout to logout from a registry.
docker search searches registry for image.
docker pull pulls an image from registry to local machine.
docker push pushes an image to the registry from local machine.

Run local registry:-

Start your registry:docker run -d -p 5000:5000 --restart=always --name registry registry:2
You can now use it with docker.

Get any image from the hub and tag it to point to your registry:

docker pull ubuntu && docker tag ubuntu localhost:5000/ubuntu

... then push it to your registry:docker push localhost:5000/ubuntu

... then pull it back from your registry:docker pull localhost:5000/ubuntu

To stop your registry, you would:docker stop registry && docker rm -v registry

Storage:-
By default, your registry data is persisted as a docker volume on the host filesystem. Properly understanding volumes is essential if you want to stick with a local filesystem storage.

Specifically, you might want to point your volume location to a specific place in order to more easily access your registry data. To do so you can:

```
docker run -d -p 5000:5000 --restart=always --name registry \
  -v `pwd`/data:/var/lib/registry \
  registry:2
```

Get a certificate

Assuming that you own the domain myregistrydomain.com, and that its DNS record points to the host where you are running your registry, you first need to get a certificate from a CA.

Create a certs directory:

```
mkdir -p certs
```
Then move and/or rename your crt file to: certs/domain.crt, and your key file to: certs/domain.key.

Make sure you stopped your registry from the previous steps, then start your registry again with TLS enabled:

```
docker run -d -p 5000:5000 --restart=always --name registry \
  -v `pwd`/certs:/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  registry:2
```
You should now be able to access your registry from another docker host:

```
docker pull ubuntu
docker tag ubuntu myregistrydomain.com:5000/ubuntu
docker push myregistrydomain.com:5000/ubuntu
```

docker pull myregistrydomain.com:5000/ubuntu
Gotcha

A certificate issuer may supply you with an intermediate certificate. In this case, you must combine your certificate with the intermediate's to form a certificate bundle. You can do this using the cat command:

cat domain.crt intermediate-certificates.pem > certs/domain.crt
Let's Encrypt

The registry supports using Let's Encrypt to automatically obtain a browser-trusted certificate. For more information on Let's Encrypt, see https://letsencrypt.org/how-it-works/ and the relevant section of the registry configuration.

Alternatives

While rarely advisable, you may want to use self-signed certificates instead, or use your registry in an insecure fashion. You will find instructions here.

Deploying a plain HTTP registry:-

https://github.com/docker/docker.github.io/blob/master/registry/insecure.md

---

Restricting access

Except for registries running on secure local networks, registries should always implement access restrictions.

Native basic auth

The simplest way to achieve access restriction is through basic authentication (this is very similar to other web servers' basic authentication mechanism).

Warning: You cannot use authentication with an insecure registry. You have to configure TLS first for this to work.
{:.warning}
First create a password file with one entry for the user "testuser", with password "testpassword":

```
mkdir auth
docker run --entrypoint htpasswd registry:2 -Bbn testuser testpassword > auth/htpasswd
```

Make sure you stopped your registry from the previous step, then start it again:

```
docker run -d -p 5000:5000 --restart=always --name registry \
  -v `pwd`/auth:/auth \
  -e "REGISTRY_AUTH=htpasswd" \
  -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
  -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
  -v `pwd`/certs:/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  registry:2
```

You should now be able to:

docker login myregistrydomain.com:5000
And then push and pull images as an authenticated user.

Managing with Compose

As your registry configuration grows more complex, dealing with it can quickly become tedious.

It's highly recommended to use Docker Compose to facilitate operating your registry.

Here is a simple docker-compose.yml example that condenses everything explained so far:

```
registry:
  restart: always
  image: registry:2
  ports:
    - 5000:5000
  environment:
    REGISTRY_HTTP_TLS_CERTIFICATE: /certs/domain.crt
    REGISTRY_HTTP_TLS_KEY: /certs/domain.key
    REGISTRY_AUTH: htpasswd
```

```
    REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
    REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
  volumes:
    - /path/data:/var/lib/registry
    - /path/certs:/certs
    - /path/auth:/auth
```

Warning: replace /path by whatever directory that holds your certs and auth folder from above. {:.warning}
You can then start your registry with a simple

```
docker-compose up -d
```

Exposing ports

Exposing incoming ports through the host container is fiddly but doable.

This is done by mapping the container port to the host port (only using localhost interface) using -p:

```
docker run -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT --name CONTAINER -t someimage
```

You can tell Docker that the container listens on the specified network ports at runtime by using EXPOSE:

```
EXPOSE <CONTAINERPORT>
```

Note that EXPOSE does not expose the port itself -- only -p will do that. To expose the container's port on your localhost's port:

```
iptables -t nat -A DOCKER -p tcp --dport <LOCALHOSTPORT> -j DNAT --to-destination <CONTAINERIP>:<PORT>
```

If you're running Docker in Virtualbox, you then need to forward the port there as well, using forwarded_port. Define a range of ports in your Vagrantfile like this so you can dynamically map them:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  ...

  (49000..49900).each do |port|
    config.vm.network :forwarded_port, :host => port, :guest => port
  end

  ...
end
```

If you forget what you mapped the port to on the host container, use docker port to show it:

```
docker port CONTAINER $CONTAINERPORT
```

Docker Security :-

Turn off interprocess communication with:

```
docker -d --icc=false --iptables
```

Set the container to be read-only:

```
docker run --read-only
```

Verify images with a hashsum:

```
docker pull debian@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Set volumes to be read only:

```
docker run -v $(pwd)/secrets:/secrets:ro debian
```

Define and run a user in your Dockerfile so you don't run as root inside the container:

```
RUN groupadd -r user && useradd -r -g user user
USER user
```