

Docker

Sunday, February 19, 2023

9:28 AM

Docker Resources

<https://www.mankier.com/1/docker-run>

<https://dockerlabs.collabnix.com/>

Docker Engine vs Docker Desktop

Docker Desktop has more... Docker Engine is open source

Sometimes you may exceed the amount of image pulls

If you login, Docker lets you pull more... you can pay and get unlimited pulls

Controlling daemon locally

You don't have to literally work on your remote machine, you can work locally

Sometimes you want to do all of your work on your local machine

You used to only have TCP connection to connect to the daemon, and it is unauthenticated

Now - can do an SSH connection if you have authentication to the server, and you don't have public ports

Brew install docker only installs the client - Mac kernels don't support containers, and on command line

Instructions for installing client only - <https://docs.docker.com/engine/install/binaries/#install-and-client-binaries-on-windows>

You can just install Docker Desktop if you want

Need to set a DOCKER_HOST env variable

Tells Docker CLI which server to talk to!

<https://code.visualstudio.com/docs/remote/ssh-tutorial>

Docker Daemon

Docker Daemon running in background

Management commands:

Docker <management command> <command>

Docker container run vs docker run

Docker Container Run

Docker container run nginx

--publish 80:80

ed
have to have

ly installs the

[install-server-](#)

Opens port 80 on host IP (left), routes traffic to the container IP on port 80 of container

--detach, -d
Lets it run in the background

--name, -n <name>
Specify a name for the container

--rm
Automatically remove container upon exiting

--entrypoint
Lets you overwrite entrypoint of image

--link=<container>
Lets you create manual link between two containers in the default bridge vnet

--network <network>
Let's you connect container to vnet upon startup

--net-alias
Allows you set a net alias in the container creation

Container Inspection commands:

docker container top <container>
Shows you basic process info

docker container inspect <container>
Shows you details configuration of the container - but not like the running container

docker container stats
Gives you streaming view about all live containers

Docker images - view all images

Docker container ls
Lists the running containers

Docker container ls -a
Shows the containers that have been stopped

Docker ps - see all containers spun up (processes)

docker network ls
Show networks

docker network inspect
inspect a network... or a container

Actions

Docker container rm 63f 690 ... -f (force delete running container)
Let's you remove deleted containers

ainer (right)

er info

Docker Connect

`docker network connect <network> <container>`

`docker run -itd --network=<network> <image>`

Automatically connect to network upon creating container

`--ip <ip>`

Specify IP address assigned to container interface

`--link <container1> <network> <container2>`

Connects two containers to the same subnet

`--alias <alias1> --alias <alias2> <network> <container>`

Connects container to subnet under an alias, so that anything trying to find can call containers with the same alias namespace

Allows you to have multiple containers that can be accessed on one call, since containers can have duplicate names

Routing

`sudo docker container port webhost`

Shows the port mappings

`docker container inspect --format {{config file thing}}`

Formats output of commands using Go Templates

`docker container inspect --format'{{ .NetworkSettings.IPAddress }}' webhost`

`docker network create --driver`

Create a network

`--network`

`docker network connect <network> <container>`

attach network to container

or can do `docker run.... --network my_app_net`

(creates NIC in container on existing virtual network)

NIC - enables computer to connect to network

Can be attached to TWO containers

detach network from container

`docker network disconnect`

Routing

Each container is connected to private virtual network "bridge"

When you start a container - you are connecting to a Docker network, the bridge network

These networks can be routed out through the NAT firewall on the host IP, which is done by the `iptables` daemon

Don't have to use `-p` (assign a port on the container) when containers are just talking to each other on the same virtual network

multiple

ainers cannot

k
by the Docker

each other on

...they, on the same virtual network.

Best practice is to create a new virtual network for every app

Non-Default adjustments:

- Make new virtual networks

- Attach containers to multiple virtual networks

- Skip virtual networks and use host IP

- Use different Docker network drivers

Not always the case that the IP address is the VM host

Anything coming out from container is going to be NATTED, and a port has to be exposed
bridge/docker0 is default to virtual network, container is attached to this vnet, and this vnet is
attached to the host

- Telling it -p 80:80 opens up 80 on mac, and to route everything from port 80 through
the container, on the container's exposed port (80)

- Another container, by default can talk to the other container on its exposed port by
running docker network inspect bridge:

- shows that the subnet IP defaults to 172.17.0.0/16, and the gateway

default networks:

- bridge - subnet

- host - skips virtual network, but sacrifices security of subnet, high throughput though

- none - just isn't attached to anything

Network driver - built in or 3rd party extensions that give you vnet features, creates virtual network
locally with its own subnet

- subnet - logical partitioned piece of a larger IP network (range of IP addresses in virtual network
vnet is divided into multiple subnets)

- IP - method of sending data from one computer to another over internet

- Default driver is bridge

DNS Resolution

Docker defaults the hostname to the container's name

docker exec -it my_nginx ping webhost

- If we attached these to the same virtual network, then this ping command should allow
two containers to interact with each other

- "webhost" and "my_nginx" are automatically made the DNS namespace, allowing the
command to work

"bridge" does not have a DNS server in there by default

-- link when creating a container allows you to make a manual link in a default bridge network
better to just create a default bridge network

- Docker compose makes automatic links between containers

"DNS" is better than using IPs for linking containers through custom networks

d
net is

gh the VNet to

y default

ugh

al network

rtual network,

allow these

chis ping

work... but its

Running Commands in a Running Container:

Docker-exec command

<https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/docker-mysql-getting-started.html>

`Docker exec --interactive -i --tty -t container_name /bin/bash`

Runs interactive shell in a running container

`docker exec --detach container_name command`

Runs command in background of running container

-t, tty - simulates a real terminal like what ssh does

-i, -interactive - keeps session open and allows us to execute more commands

`docker container run -it <Image> bash`

bash will take you to a bash shell and give you a bash to work with

If you do `ls -al`, it will literally show you the file directory structure of the container

if you run `docker container ls`, it will show the start command to be bash

`docker container run -it --name ubuntu ubuntu`

would download the whole ubuntu image to the container (don't have to specify shell as default)

This will be a much more minimal version of ubuntu

when you exit the shell, it stops the container

`docker container start -ai`

start previously existing container that had a shell attached, -a attaches STDOUT and STDERR

`docker container exec -it <container> bash`

Runs an ADDITIONAL interactive process (or I guess any process) inside a RUNNING container with a shell

`docker container exec -it mysql bash`

Puts you in interactive bash on container where mysql is running

Run `ps` :

Will show you the mysql process

AND will show you the bash process being ran in the container

When you exit, mysql is still running. That is because the bash script is an ADDITIONAL process

Alpine!

Very very small linux distro (only 5 mg)

can do `docker pull alpine`

Comes with package manager - so you can download stuff on alpine

`docker container run -it alpine bash`

error... because it does not even have bash in it

need to use "sh"

Can use `apk` to install stuff

[ted.html](#)

hell bc that is

r something

g container

NAL process

What happens when we run a container?

- 1 Looks for image locally in image cache
- 2 If it doesn't find, it defaults Docker Hub to check
- 3 Will store latest version by default
- 4 Creates new container based on that image and prepares it to start
- 5 Gives it new virtual IP on a private network inside the docker engine
- 6 Opens up port 80 on the host and forwards to virtual port 80 on the container
- 7 starts container using the CMD in the image Dockerfile

Mini Assignment:

Running and connecting nginx, mysql, and httpd server

Httpd: HTTPD server daemon which listens for network requests and responds to them, server is Apache Tomcat

Nginx: Asynchronous approach for handling web requests, similar to running httpd or ap

MySQL Access:

<https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/docker-mysql-getting-started.html>

Container Vs VM:

Containers are not VMs: They are just processes, are limited to what resources they can access, they exit when the process stops

Container Images

Commands:

docker history nginx:latest

What is an image?

App binaries and dependencies

Metadata about image data and how to run the image

Not the complete OS! No kernel or kernel modules (drivers)

Can be small as a static binary or big as an Ubuntu distro with apt, Apache, PHP and more

Image Layers

similar HTTP

oache

[ted.html](#)

access, and

e installed

Images are stacked in layers, and the same layers can be reused in other images
Containers don't have to download the same layer more than once... I don't think
"Copy on write" - containers are reusing the same code from the image - when one container
code or a file, then it will copy the new file and place it in that container replacing the image
In the image list (history), you can see as the image layers have gotten changed over time
layers are not assigned an id since they are just layers inside of the image

Azure Container Registry

Push images to the ACR - ACR allows you to host your images, like DockerHub

Can download Image onto your VM, and run it there

But, if you delete your VM, you can run Azure Container Instances to run these containers
VM

- Create a "Container Resource" instance

- Can deploy

Azure Container Groups

- Use Azure CLI commands to deploy an "Azure Container Group"

- YAML config for app deployment details what a container is (name of image and repository,
number of compute resource to have, what port to expose, what os Type, assigning
imageRegistry login details)

- Can upload the app yaml file directly which creates the container instance

ainer changes
age code
e, but the

s without a

gistry,
of public,