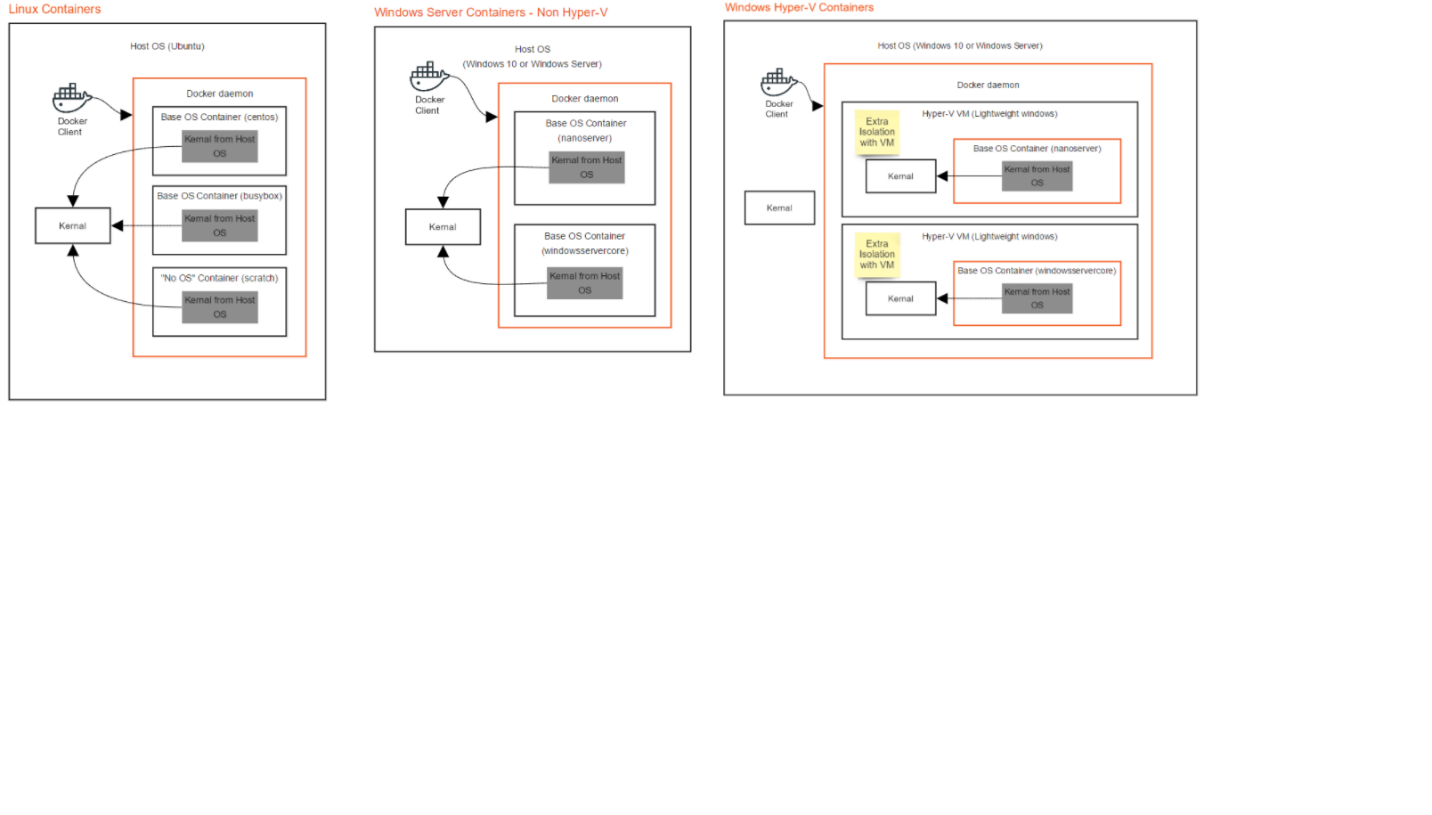
Docker notes:

Some concepts:

Sometimes it’s easily confused on the difference among VM, Vagrant and Docker container. 1. VM is just virtual machine which has many tools like oracle virtualbox ect. 2. Vagrant is the tool to help us automatically setup VM and related tools in VM by vagrantfile.yml. Vagrant must be applied on host machine. It cannot execute on VM because itself will create a VM (second level of VM always has problem). 3. Docker is a container. It has its own base OS. The base OS communicates to host OS through docker engine(Kernel). This [article](http://floydhilton.com/docker/2017/03/31/Docker-ContainerHost-vs-ContainerOS-Linux-Windows.html) perfectly explained the related concepts on docker.



. install docker (<https://docs.docker.com/get-started/#setup> )

2 ways to install docker. 1. Install docker from repository. Before installing docker, repository must be added into OS repository list.

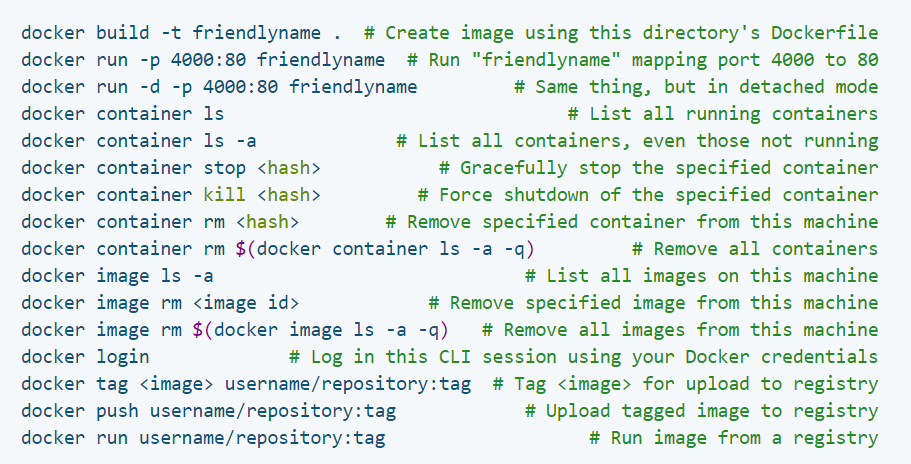
Debian and Centos have different approaches to manage their repositories (details see ref above). In Ubuntu, must download public key before download package from repository, which is used during download. The key is managed by apt-key (gpg—GNU Private Guard). Repository is stored in /etc/apt/sources.list file. In Centos, repository is managed by yum-config-manager –add-repo url. 2. The other approach is just to download the package and install the package.

. Define docker container with Dockerfile

Dockerfile syntax <https://codefresh.io/docker-guides/build-docker-image-dockerfiles/>

Dockerfile defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you have to map ports to the outside world, and be specific about what files you want to “copy in” to that environment. However, after doing that, you can expect that the build of your app defined in this Dockerfile will behave exactly the same wherever it runs.

Below are the common docker commands: ([docker commands](https://docs.docker.com/engine/reference/commandline/commit/))



$>docker run -d -it -v data:/shared --name containername imagefile bash #create and start container from image. The volume data is automatically created and mount at /shared folder in container. Where the folder locate in host machine can be found by

$>docker volume ls #show all created volumes

$>docker volumn inspect data # show the details of the volume whose name is data

We can also mount the absolute directly in host machine into container by

$> docker run -d -it -v /home/simon/data:/shared --name containername imagefile #mount absolute folder. Absolute folder will not be managed by docker. It means you cannot find the volume from ‘docker volume ls’

$>docker exec -it containername bash #log into the container with bash ssh session

. manually create docker image

From last session of Dockerfile, it explains how to automatically build an image and push into docker hub. In real case, we might need to manually to approve each step defined in Dockerfile. For the case of this, below is the steps to do that:

Before looking the details, we must clear the difference of these 2 concepts of image and container. Container is the instance of image. It means one image can have multiple containers.

1. Make sure docker daemon is running

$>service docker status #or

$> ps awx | grep docker

$>service docker start #to start docker engine

$>docker #get help of docker commands

$>docker info #get docker information

$>docker search postgres #show available images for postgresql

1. Docker pull to download base image we like to build upon
   1. $>docker pull postgres
   2. $>docker run –d –it –name mypostgrescontainer postgres #here postgres is image name

I didn’t see any help docker create. Docker run can create a container from image and start running it. Docker start can only start the container which is stopped by command of docker stopped

docker run --name postgres953 -e POSTGRES\_PASSWORD=postgres -p 5432:5432

1. Docker run to start the container
   1. $>docker run -d -i -t mypostgrescontainer postgres
   2. $>docker exec -it mypostgrescontainer bash #log into container with bash ssh

. Docker volume (<http://container-solutions.com/understanding-volumes-docker/> )

Docker volume is the approach to share the data among containers or between container and host machine. The most common way to create volume is during creating and starting a container ie. $>docker run -d -it -v data:/shared --name containername imagefile bash

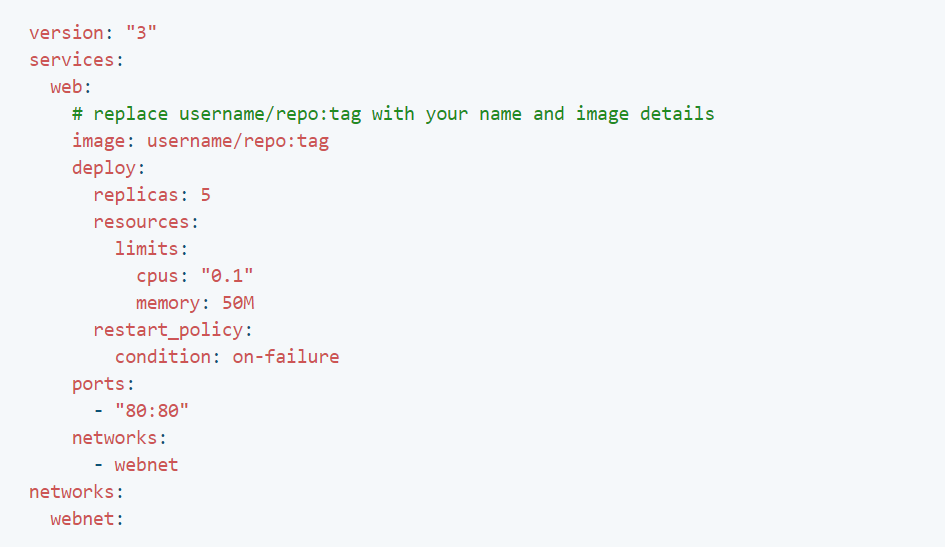
The command above is to create and start a new container (which is named as containername) from imagefile by detached and interactive mode and run bash command immediately.

Use case of docker volume

HOW TO PERSIST THE CONTAINER DATA.

. Docker Service

When application is ready to production, we might consider about startup multiple instances of apps and load balance and. Docker swarm can naturally support that by defining docker-compose.yml file. See the sample file below:



This docker-compose.yml file tells Docker to do the following:

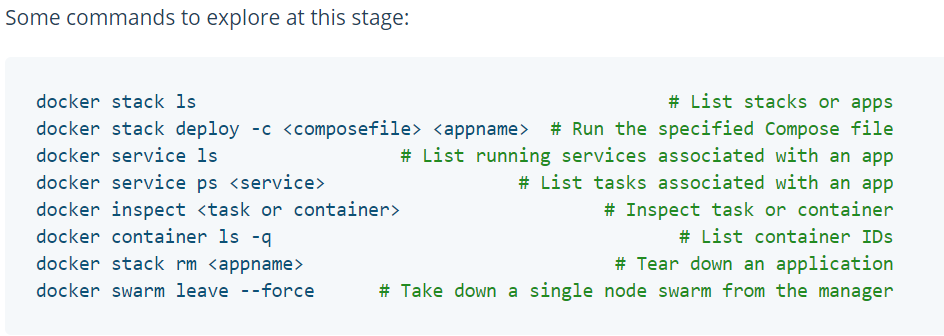
* Pull [the image](https://docs.docker.com/get-started/part2/) from the registry.
* Run 5 instances of that image as a service called web, limiting each one to use, at most, 10% of the CPU (across all cores), and 50MB of RAM.
* Immediately restart containers if one fails.
* Map port 80 on the host to web’s port 80.
* Instruct web’s containers to share port 80 via a load-balanced network called webnet. (Internally, the containers themselves will publish to web’s port 80 at an ephemeral port.)
* Define the webnet network with the default settings (which is a load-balanced overlay network).

Some related concepts:

Service: it is distributed application. One service runs as one image file. One service stack can run multiple application services in multiple replicas containers, which is defined in docker-compose.yml file.

Task: a single container running in a service is called a task. Tasks are given unique IDs.

The commands below shows how to deploy application to stack.



. Docker Swarm

Swarm is the working mode integrated into Docker Engine since 1.12 and has brought with several new tools. Now docker compose tool is no longer needed to run docker swarm. It has been replaced by docker stack in the new version.

In the above, we replicated 5 our application instances in the same host machine by using docker service. Here, we gonna introduce docker swarm which allows the containers to run on multiple hosts? in cluster env. The tool of building docker cluster env is docker-machine. So the first step of docker swarm is to install docker machine.

curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&

chmod +x /tmp/docker-machine &&

sudo cp /tmp/docker-machine /usr/local/bin/docker-machine

. verify of installation: ‘docker-machine version’

. swarm manager and workers: swarm manager are the only machine in a swarm that can execute the commands or authorize other machines to join the swarm as workers. Workers are just there to provide capacity and do not have the authority to tell any other machine what it can and cannot do.

$>docker swarm init #start docker machine as manager

$>docker swarm join #docker worker join the swarm