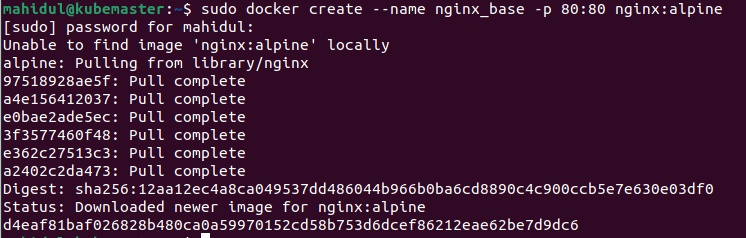
This container will be created/provided by your developer.

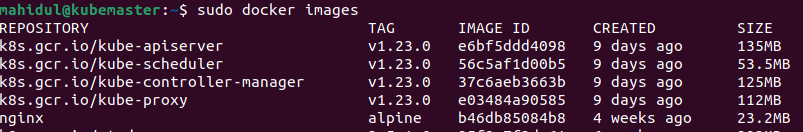
**Step 1: Create a Base Container**

developer@pc:~$ sudo docker create --name nginx\_base -p 80:80 nginx:alpine



**Step 2: Inspect Images**

developer@pc:~$ sudo docker images



**Step 3: Inspect Containers**

developer@pc:~$ sudo docker ps -a

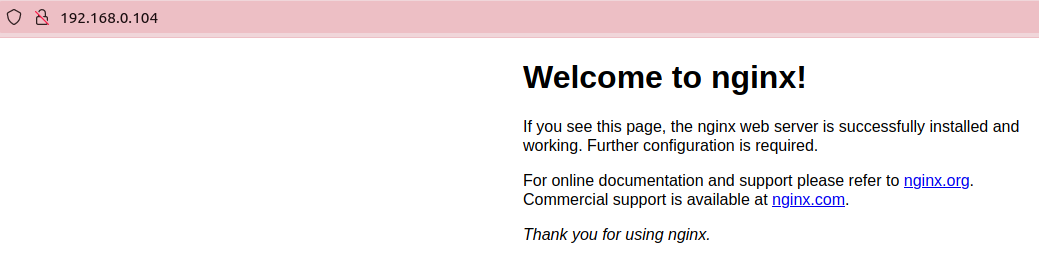


**Step 4: Start the Container**

developer@pc:~$ sudo docker start nginx\_base

Check the status

<http://localhost>



**Step 5: Modify the Running Container**

developer@pc:~$ vim index.html

<html>

<head>

<title>Hello world !!!</title>

</head>

<body>

<h1>Hello world !!!</h1>

</body>

Copy the web page to the container

developer@pc:~$ docker cp index.html nginx\_base:/usr/share/nginx/html/index.html

Check the status

<http://localhost>



**Step 6: Create an Image From a Container**

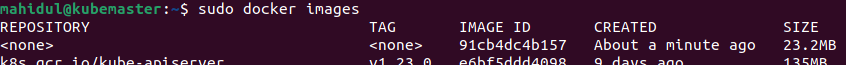
Think of an image like an .iso file. It's not an active container/vm box.

developer@pc:~$ sudo docker commit nginx\_base

Sha256:91cb4dc4b15776a0fa1122953ae93a76cb1fef2844a985074b85bd6c23b07b43



developer@pc:~$ sudo docker images



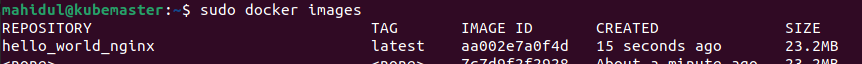
**Step 7: Tag the Image**

developer@pc:~$ sudo docker tag 91cb4dc4b157 hello\_world\_nginx

**Step 8: Create Images With Tags**

developer@pc:~$ sudo docker commit nginx\_base hello\_world\_nginx

developer@pc:~$ sudo docker images



**Step 9: Delete the Old/Original Container**

developer@pc:~$ sudo docker ps



developer@pc:~$ sudo docker stop nginx\_base

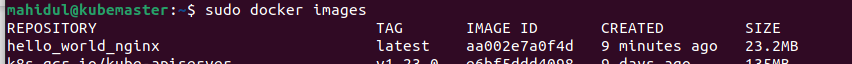
nginx\_base

developer@pc:~$ sudo docker rm nginx\_base

nginx\_base

**Step 10: Run an image**

developer@pc:~$ sudo docker images

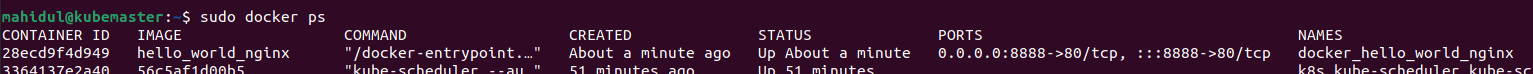


developer@pc:~$ sudo docker run --name docker\_hello\_world\_nginx -d -p 8888:80 hello\_world\_nginx



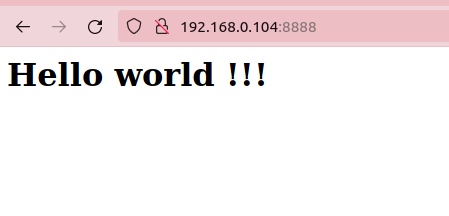
The -rm will cause the container to be deleted when it is shut down. The -d tells the command line client to run in detached mode. This will allow us to run other commands from the same terminal.

Look at Running Containers



Check the web page

http://localhost:8888/



**Step 10: Copy an image to the production environment (*How to copy Docker images from one host to another without using a repository*)**

***Option#1: Save and load images***

We have to save the Docker image as a tar file:

developer@pc:~$ docker save -o <path for generated tar file> <image name>

Then copy your image to a new system with regular file transfer tools such as cp, scp or rsync(preferred for big files). After that you will have to load the image into Docker:

developer@pc:~$ docker load -i <path to image tar file>

***Option#2: Export and import containers***

Export a docker image

developer@pc:~$ docker export container-name | gzip > container-name.gz

Import a docker image

developer@pc:~$ zcat container-name.gz | docker import - container-name

***Option#3 How ro deploy image at AWS***

#host1

systemctl stop docker

systemctl start docker

docker commit -p 1d09068ef111 ubuntu001\_bkp3

#create backup

docker save -o ubuntu001\_bkp3.tar ubuntu001\_bkp3

#upload ubuntu001\_bkp3.tar to my online drive

aws s3 cp ubuntu001\_bkp3.tar s3://mybucket001/

#host2

systemctl stop docker

systemctl start docker

cd /dir1

#download ubuntu001\_bkp3.tar from my online drive

aws s3 cp s3://mybucket001/ubuntu001\_bkp3.tar /dir1

#restore backup

cat ./ubuntu001\_bkp3.tar | docker load

docker run --name ubuntu001 -it ubuntu001\_bkp3:latest bash

docker ps -a

docker attach ubuntu001

**Step 10: If you are managing container with Kubernetes then follow the below instructions**

After copying the image run it with kubernetes

mahidul@kubemaster:~$ kubectl run podhelloworld --image=hello\_world\_nginx --restart=Never

****

***Active Kubernetes load balancer***

i. we will be leveraging Rancher's k3d to run a local Kubernetes cluster and

ii. installing MetalLB as a Load Balancer to our cluster.

Prerequisites

Docker

k3d (v4.4.6)

jq

kubectl

lens (optional)

setup

# create the k3d cluster

k3d cluster create local-k8s --servers 1 --agents 3 --k3s-server-arg --no-deploy=traefik --wait

# set kubeconfig to access the k8s context

export KUBECONFIG=$(k3d kubeconfig write local-k8s)

# validate the cluster master and worker nodes

kubectl get nodes

Determine your Load Balancer's ingress range by obtaining it's cidr block. This range will depend on the Docker network that your k3d cluster leverages. The script below will help determine and prescribe a suggested range.

# determine loadbalancer ingress range

cidr\_block=$(docker network inspect k3d-local-k8s | jq '.[0].IPAM.Config[0].Subnet' | tr -d '"')

cidr\_base\_addr=${cidr\_block%???}

ingress\_first\_addr=$(echo $cidr\_base\_addr | awk -F'.' '{print $1,$2,255,0}' OFS='.')

ingress\_last\_addr=$(echo $cidr\_base\_addr | awk -F'.' '{print $1,$2,255,255}' OFS='.')

ingress\_range=$ingress\_first\_addr-$ingress\_last\_addr

Deploy the Load Balancer, which leverages MetalLB:

# deploy metallb

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.10.2/manifests/namespace.yaml

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.10.2/manifests/metallb.yaml

# configure metallb ingress address range

cat <<EOF | kubectl apply -f -

apiVersion: v1

kind: ConfigMap

metadata:

namespace: metallb-system

name: config

data:

config: |

address-pools:

- name: default

protocol: layer2

addresses:

- $ingress\_range

EOF

Validation

# create a deployment (i.e. nginx)

kubectl create deployment nginx --image=nginx

# expose the deployments using a LoadBalancer

kubectl expose deployment nginx --port=80 --type=LoadBalancer

# obtain the ingress external ip

external\_ip=$(kubectl get svc nginx -o jsonpath='{.status.loadBalancer.ingress[0].ip}')

# test the loadbalancer external ip

curl $external\_ip

Teardown

Destroy the cluster

k3d cluster delete local-k8s