*Docker & Kubernetes*

Docker is a container management service. The keywords of Docker are **develop, ship** and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

What container includes?

1.application (amazonapp.war – java app)

2.depedencies(odjc6.jar, hibernate.jar)

3.libraries(tomcat server)

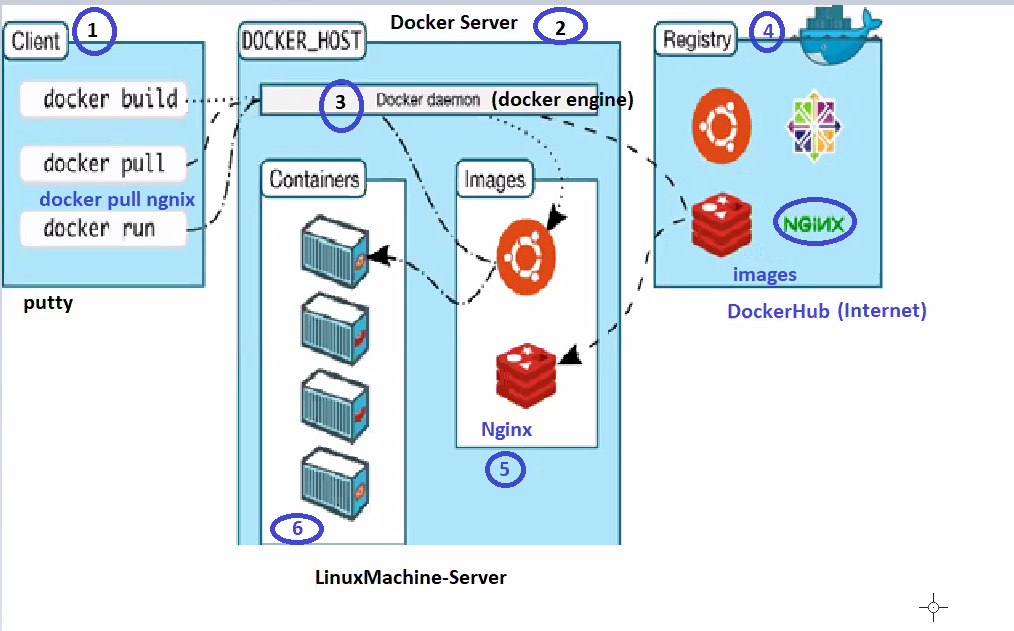
4.Binaries(tomcatserver)

5.configuration files(tomcat config)

Architecture:

It works on client – server architecture

Docker daemon (docker engine) -> will come with docker installation



Docker image: read only template with instruction for creating a docker container.

Conatiners are the instances of images – they are the living things.

Registry: a repository of image

DockerFile: a file that containing instruction to create image

Demo: lab

Docker file:

Text doc, contains all the commands a user could call on command line to assemble an image.

Docker file command:

FROM: it defines the base image to user to start the build process

It needs to be the first command declear in docker file

MAINTAINER: this non-executing command to declar author name

RUN: run it to form the image. (build image)

USER: to pass username or user cmd

VOLUME: use to enable access from your container to a directory on the host machine.

WORKDIR: workdir directive is used to set where the command defined with CMD is to be executed.

ADD:

CMD: not execute during the run

ENV: to set the environment variables

ENV JAVA\_HOME-/opt/java1.8

BUILD: build an image from dockerfile

**Lab1- Docker file**

Step-1:

vi Dockerfile

FROM centos:latest

MAINTAINER admin

RUN yum install -y httpd

COPY index.html /var/www/html/

EXPOSE 80

CMD["/usr/sbin/httpd","-D","FOREGROUND"]

Esc : wq

Step-2: creae a index.html file

<html>

<body bgcolor="pink">

<h2>Welcome to India! </h2>

</body>

</html>

step-3: using docker build command, we need to create a custom image

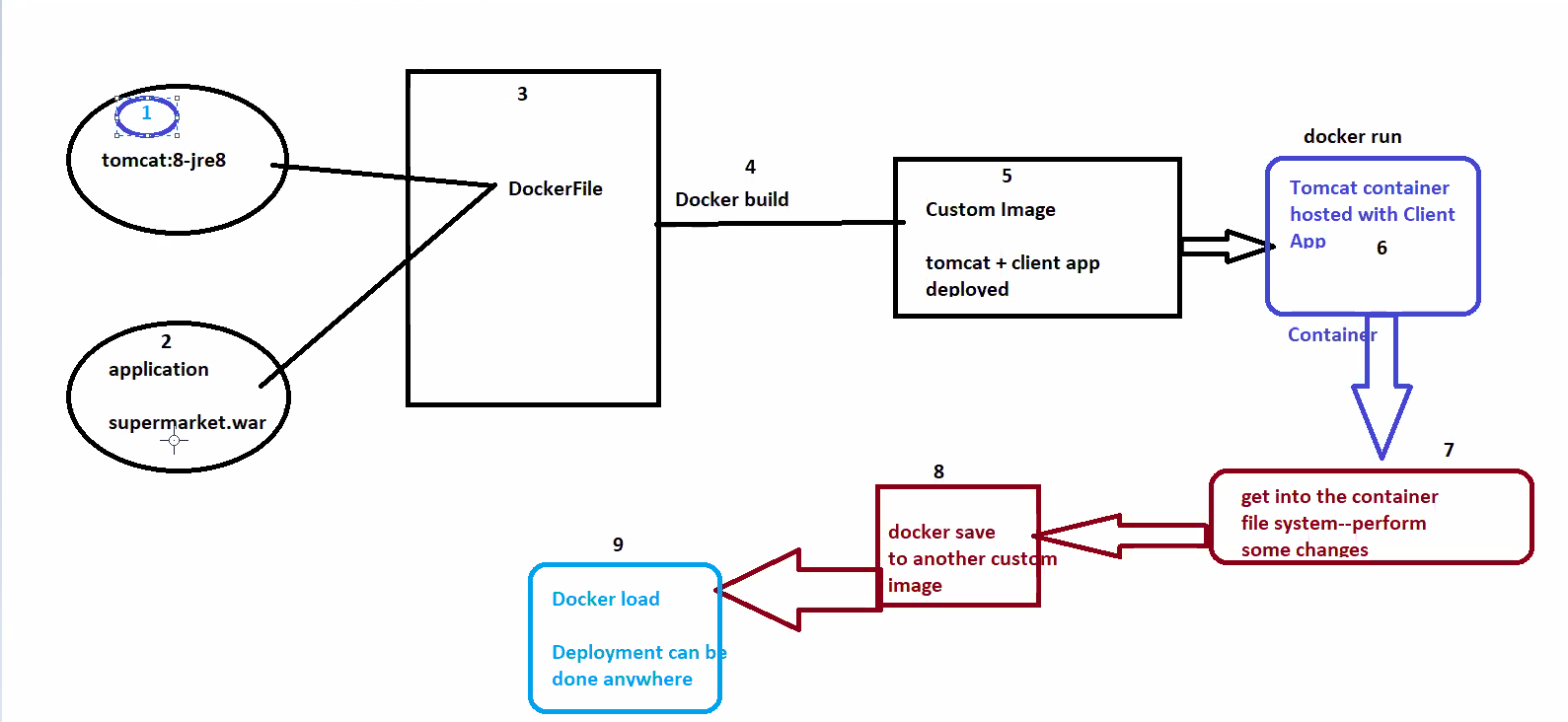
Syntax: docker build -t <imagename> .

docker build -t mycentos .

Step-4: start the container using this custom images

docker run -d --name <containername> <imagename>

docker run -d --name myapachec1 mycentos



Building/Deploying Java Webapplication using tomcat from Github to docker

Step-1:

mkdir -p /opt/dockerlabs/

git clone https://github.com/devopstraining4/tomcatsupermarket.git

Step-2:

docker build -t mytomcatapp .

step-3:

docker run -d --rm -p 8082:8080 --name supermarketc1 mytomcatapp

step-4:

to access the app using linux host ip from windows browser

http://192.168.221.128:8082/supermarket/

to access using container ip with port

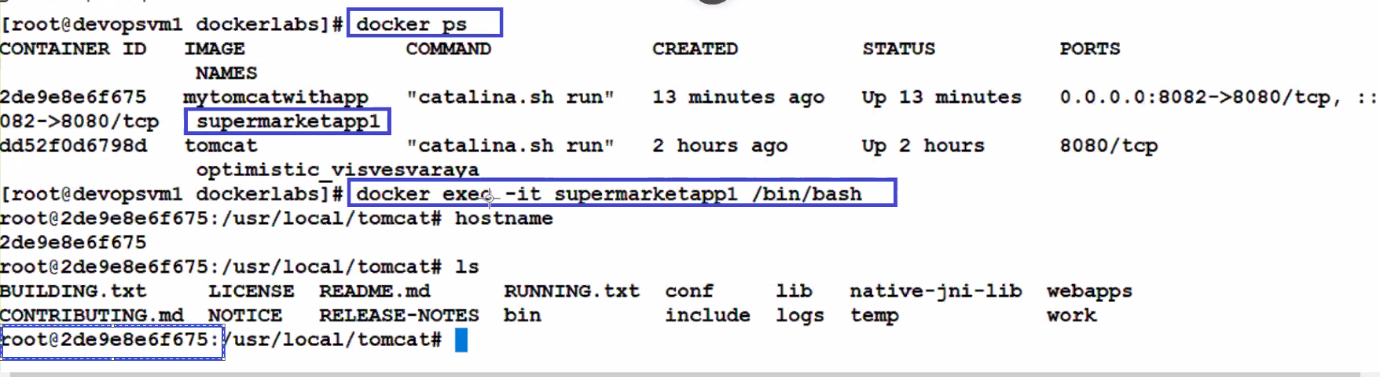
http://172.17.0.2:8080/supermarket/

docker inspect containername| grep IPA

Step-5:

to get into the container filesystem

docker exec -it <containername> /bin/bash



Step-6:

docker commit <containername-running> <new image name>

docker commit supermarketapp1 supermarketapp

Step-7:

docker save <imagename> > fileanme.tar

docker save supermarketapp > supermarketapp.tar

Step-8:

(before loading, for testing purpose delete the previous images)

docker load -i filename.tar

docker load -i supermarketapp.tar

Step-9:

docker run -d --name c1 supermarketapp

--------------------------------------------------------------------------------------------------------

To delete docker image:

* docker rmi image-name

To stop the container

* docker stop containername

--------------------------------------------------------------------------------------------------------

Volumes:

Persisting data generated by and used by Docker containers.

1. Volumes
2. Bind mounts

Commands:

* docker volume

To check existing volumes:

* docker volume ls

To create new volume

* docker volume create muvolume1

To check where the volume is created:

* docker volume inspect myvolume1

volumes data location: /var/lib/docker/volumes/myvolume1/\_data

docker run --name myjenkinsc1 -v myvolume1:/var/jenkins\_home -p 8081:8080 -p 50000:50000 jenkins/jenkins

docker logs <container name>

docker logs myjenkinsc1

starting new Container to check volume data

docker run --name myjenkinsc2 -v myvolume1:/var/jenkins\_home -p 8082:8080 -p 49000:50000 jenkins/jenkins

docker volume rm myvolume1

---------------------------

Docker bindmount

mkdir -p /opt/jenkinsbm

docker run -u root --name myjenkinsbmc1 -v /opt/jenkinsbm:/var/jenkins\_home -p 8082:8080 -p 49000:50000 jenkins/Jenkins

portainer for docker management ui

docker pull portainer/portainer

**Docker networking:**

To get all available networks:

* docker network ls

To inspect the bridge network:

* docker network inspect bridge

To check IPA of current container

* docker inspect <containername> | grep IPA

docker network create mynetwork1

docker network inspect mynetwork1

docker network ls

---------------------------------------------------------------

docker run -it --name centosc1 centos

yum install -y net-tools

yum install -y wget

docker commit centosc1 centoswithNetwork

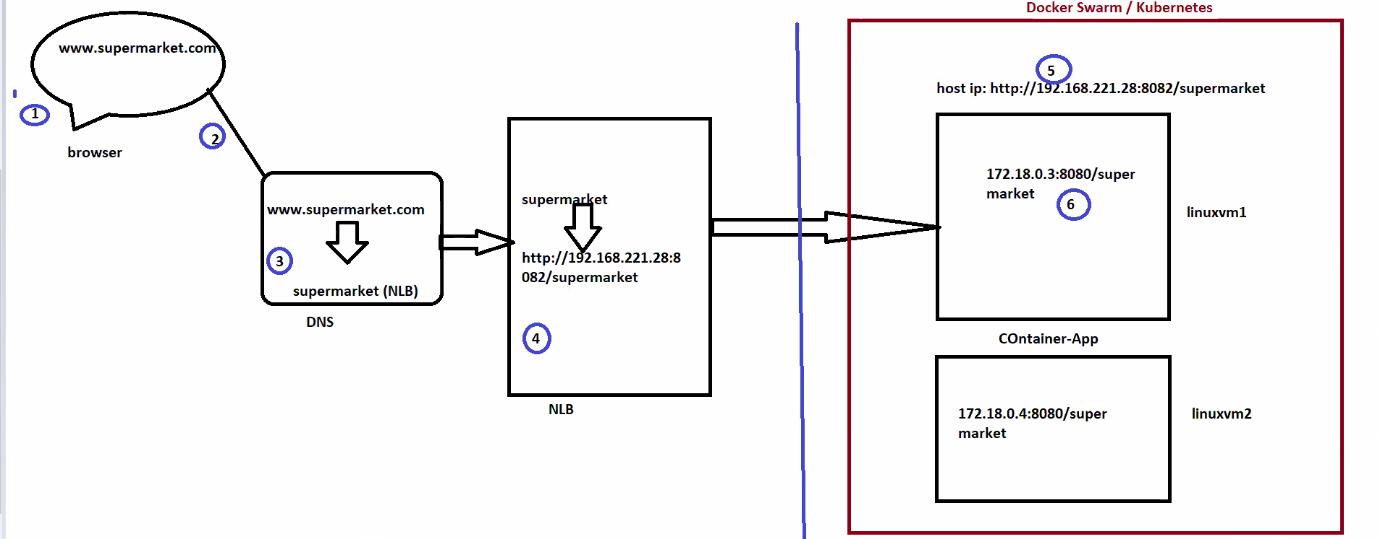
docker images

docker run -it --name mycontainern2 --network mynetwork1 centoswithnetwork

docker network create mynetwork2 --subnet 10.0.9.0/24

docker run -it --name mycontainern1 --network mynetwork1 centoswithnetwork /bin/bash

docker run --name myjenkinsc3 -v myvolume1:/var/jenkins\_home -p 8082:8080 -p 49000:50000 --network mynetwork1 jenkins/Jenkins



Example with Subnet: (Assigning subnet to docker network):

Docker network create mynetwork2 --driver overlay –subnet 10.0.9.8/24

portainer for docker management ui

docker pull portainer/portainer

docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer

* to restart docker

docker restart

**Puppert:**

Puppet is a software configuration management tool which includes its own declarative language to describe system configuration.

Linking the container

docker pull puppet/puppetserver

docker pull puppet/puppet-agent-alpine

Step-1: start the puppetserver

docker run --network mynetwork1 -d --name puppetserver --hostname puppet puppet/puppetserver

docker exec puppetserver /bin/bash

Step-2: link and start puppet agent

docker run --network mynetwork1 --link=puppetserver:puppet puppet/puppet-agent-alpine

Docker Compose:

Is used to run multiple containers as single service.

**YAML Maps**

name: value

**Yaml list:**

args ----------🡪 object name

* india
* 200
* Test

Installation of Compose

Step-1: curl -L https://github.com/docker/compose/releases/download/1.28.5/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

Step-2: chmod +x /usr/local/bin/docker-compose

Step-3: docker-compose --version

------------------------------------------------------------------------------------

cat docker-compose.yaml

version: '3.8'

services:

webservers:

image: nginx

ports:

- 8089:80/tcp

databaseservers:

image: redis

ports:

- 6379:6379/tcp

To validate the docker compose yml file:

docker-compose config

To start the docker compose file:

docker-compose up -d

docker-compose ps

docker pull bitnami/rabbitmq

5672

jboss/wildfly

9990:9990

To stop the docker compose running file:

* docker-compose down

docker hub login from cmd:

* docker login
* docker images

docker tag <imageid> <dockeruser-id>/<imagename>:<version>

docker tag 589f09a6d26d middlewaredevops4/centoswithnetwork:latest

docker push <dockeruser-id>/<imagename>:<version>

docker push middlewaredevops4/centoswithnetwork:latest

verify by logging into docker hub

**Kubernetes**

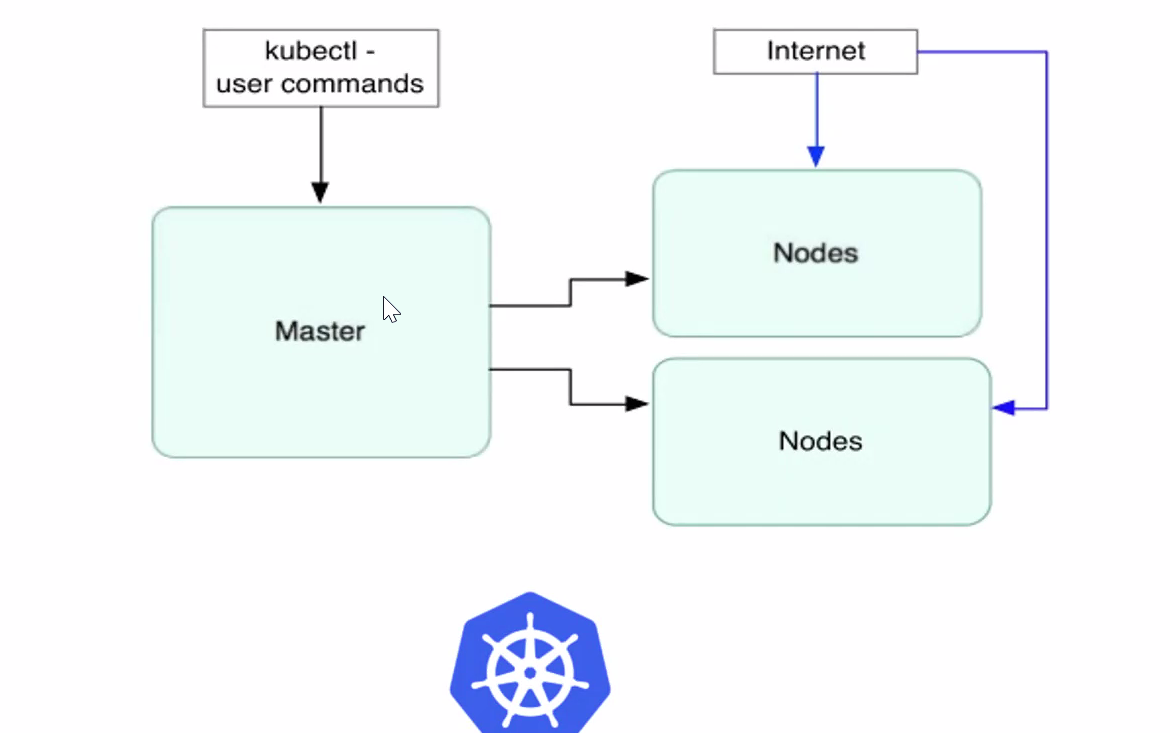
It orchestration system for docker containers

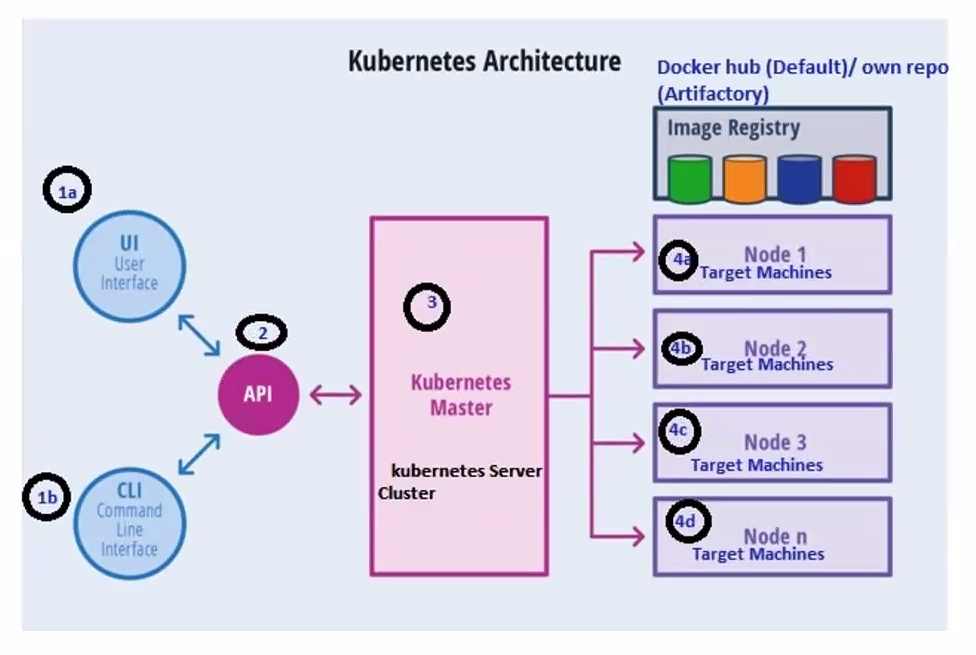
Service for container cluster management

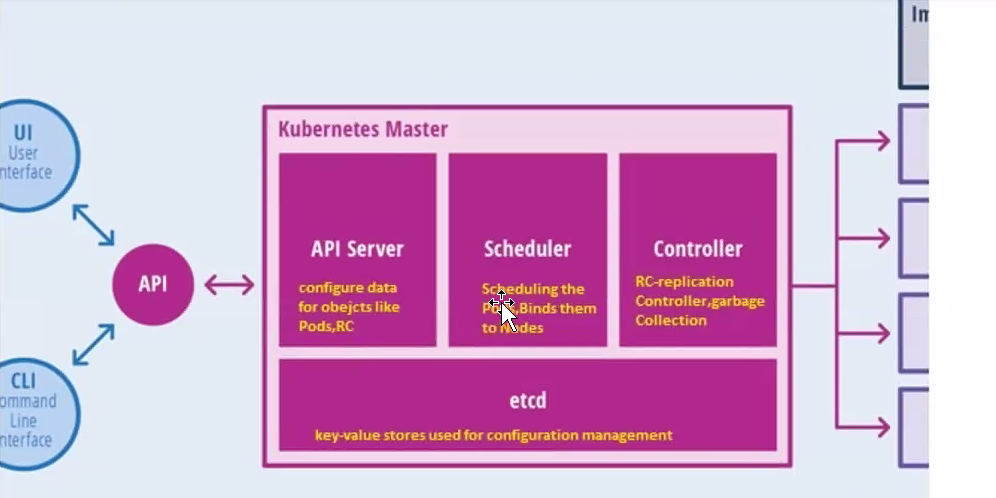
Used to manage docker container as a default implementation

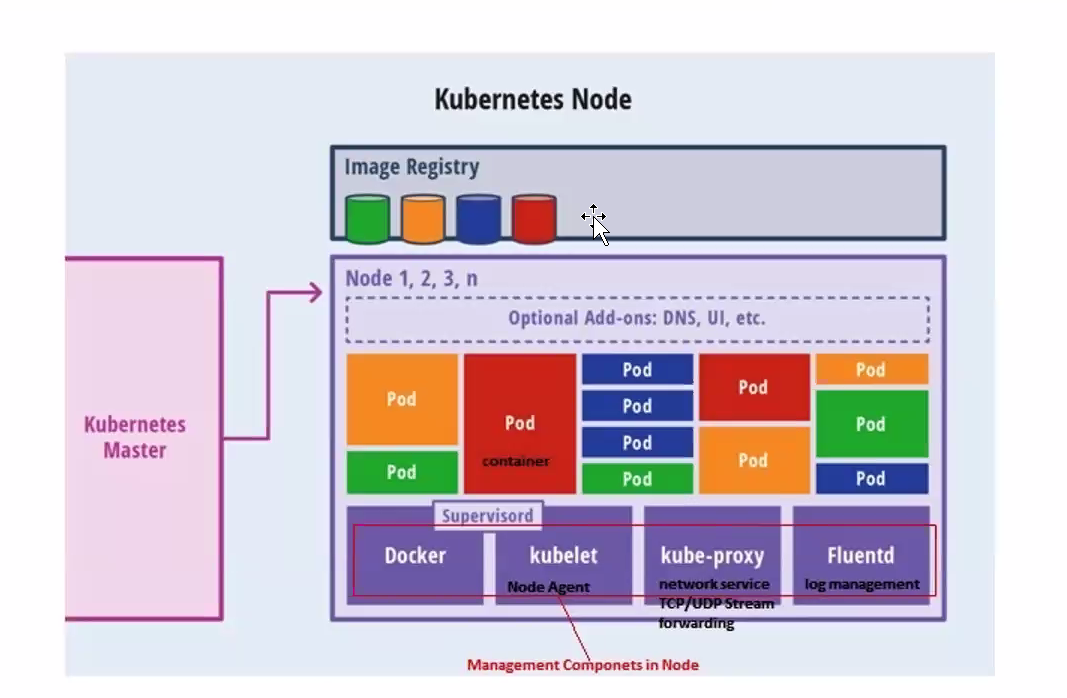
Opensource orchestration system for docker containers

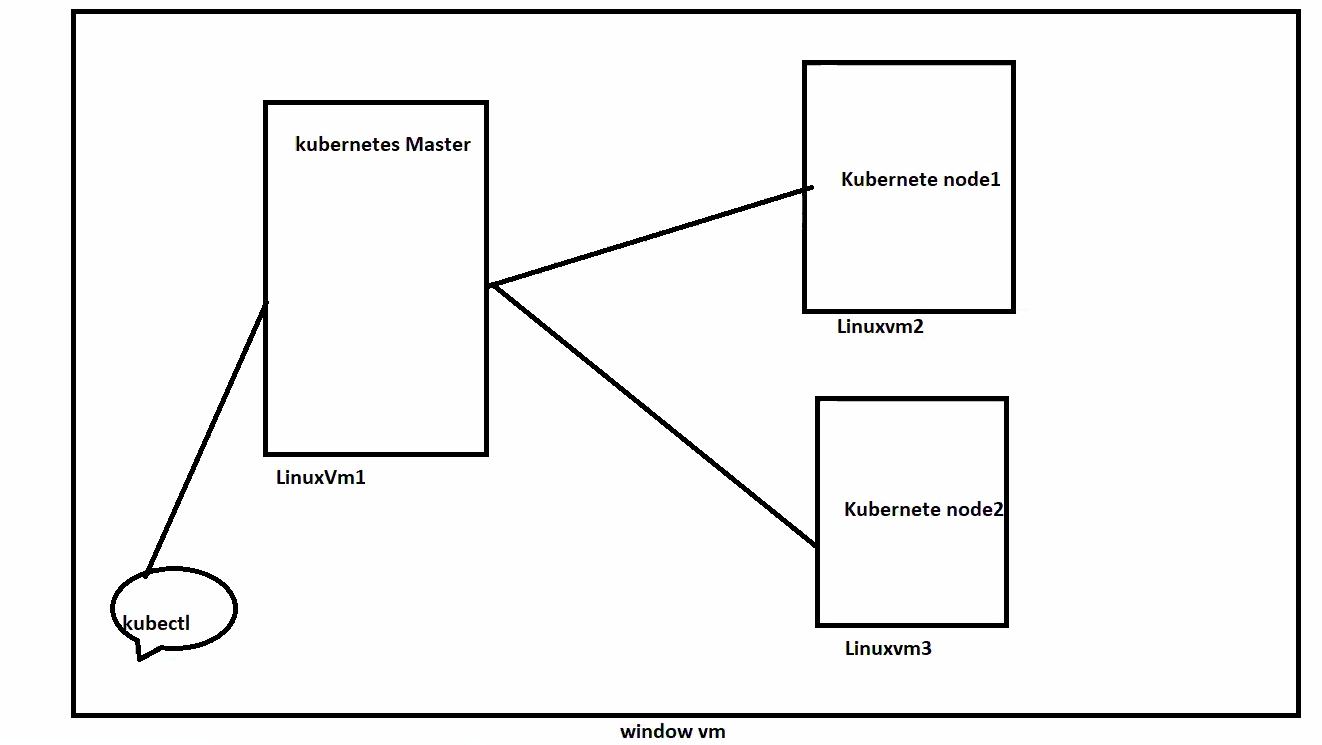
Kubernates will manage the state of those containers











Steps to setup kubernetes labs

username: root

password: redhat

Kubemaster vm 4GB Ram 2 cpu

knode1 vm 4GB RAM 1 CPU

knode2 vm 4GB RAM 1 CPU

Perform the below step in all the 3 vms

1.setup hostnames with ipaddress

vi /etc/hosts

192.168.198.188 kmaster

192.168.198.189 knode

192.168.198.190 knode2

in kmaster

hostnamectl set-hostname kmaster

in knode1

hostnamectl set-hostname knode1

in knode2

hostnamectl set-hostname knode2

2. disable the selinux and enable br\_netfilter Kernel Module (iptables for filtering,pods communication)

vi /etc/selinux/config

set to disabled

disable the firewall

systemctl status firewalld

systemctl stop firewalld

systemctl disable firewalld

systemctl status firewalld

3. set the netfilter kernel modules (iptables for filterings for pods commu

nications)

modprobe br\_netfilter

echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

4. Disable SWAP

swapoff -a

5.vi /etc/fstab (not required for Azure Linux vms)

comment

# #/dev/mapper/cl-swap

6. Install Docker CE Docker Community Edition (CE) and

package dependencies for docker-ce.

yum install -y yum-utils device-mapper-persistent-data lvm2

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

yum install docker-ce

docker -v

Note: Dont Start Docker

7. Install Kubernetes

Add the kubernetes repository to the centos 7 system by running the following command.

copy and paste the below in commandline

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg

https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

8. yum install -y kubelet kubeadm kubectl

After the installation is complete, restart all those servers.

sudo reboot

9. Log in again to the server and start the services, docker and kubelet.

systemctl start docker && systemctl enable docker

systemctl start kubelet && systemctl enable kubelet

10. Change the cgroup-driver

cgroups (control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

docker info | grep -i cgroup

Now run the command below to change the kuberetes cgroup-driver to 'cgroupfs'.

sed -i 's/cgroup-driver=systemd/cgroup-driver=cgroupfs/g' /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf

Reload the systemd system and restart the kubelet service.

systemctl daemon-reload

systemctl restart kubelet

complete the above steps in all machines( master node machines)

------------------------------------------------------------------------------------------------------------------

Step 11 - Kubernetes Cluster Initialization (in Master Machine Only)

kubeadm init --apiserver-advertise-address=192.168.198.188 --pod-network-cidr=10.244.0.0/16

example: kubeadm init --apiserver-advertise-address=192.168.187.210 (masterip) --pod-network-cidr=10.244.0.0/16 (pod network called flannel (virtual network)ip series)

Note:

--apiserver-advertise-address = determines which IP address Kubernetes should advertise its API server on.

--pod-network-cidr = specify the range of IP addresses for the pod network. We're using the 'flannel' virtual network. If you want to use another pod network such as weave-net or calico, change the range IP address

sub steps of Step-11

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

4. kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

To get the nodes:

* kubectl get nodes

Pod:

is collection of one or more nodes.

kubectl get pods --all-namespaces

kubectl run <podname> --image= imagename --port=no --generator=run-pod/v1

kubectl run mynginx1 --image=nginx --port=80 --generator=run-pod/v1

To get the status of pod with details

kubectl get pods -o wide

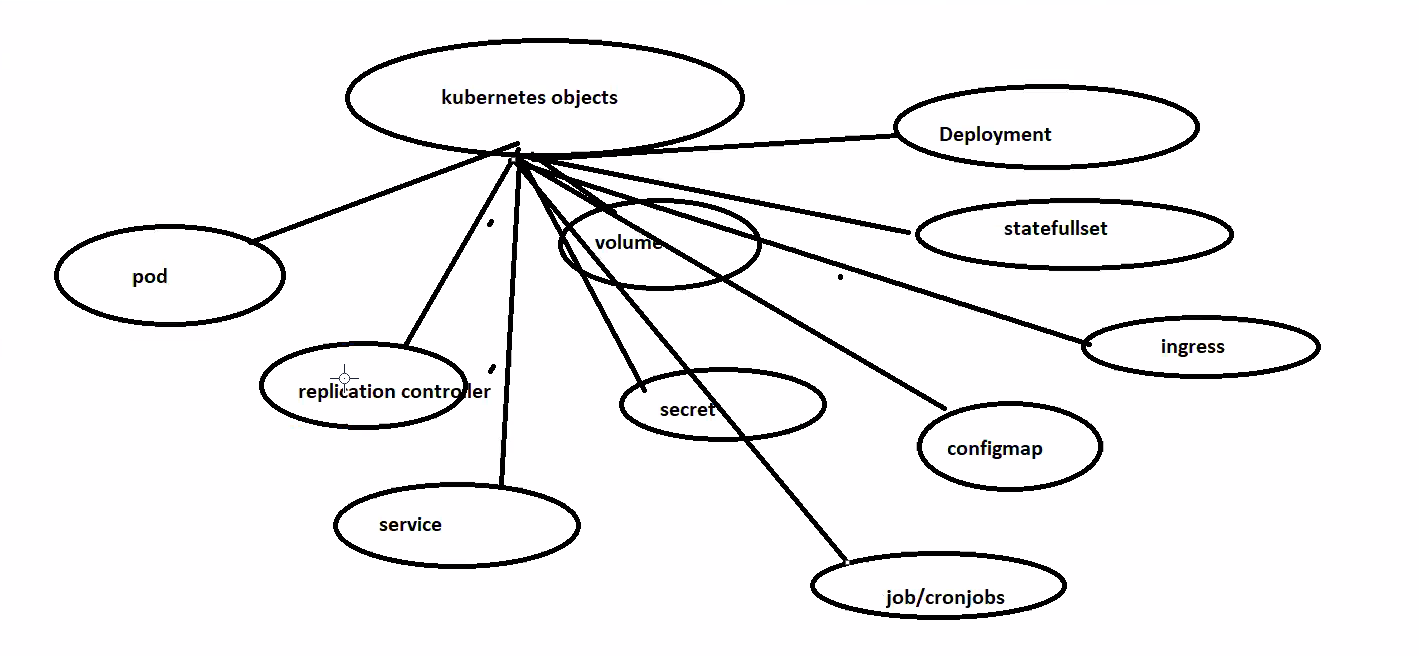
To get all the pods:

kubectl get pods

To check the pod events:

kubectl describe pod mynginx1

docs: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#pod-v1-core>



Label: key-values pairs which are attached to pods

Used as identifying attributes for objects such as pods and replication controller.

vi pod1.yml:

apiVersion: v1

kind: Pod

metadata:

name: indiapod1

labels:

name: mynginxserver

spec:

containers:

- name: indiapod1

image: nginx

ports:

- containerPort: 80

To run the yml file:

* kubectl create -f pod1.yml

To check the status:

* kubectl get pods -o wide

To check the logs:

* kubectl logs indiapod1

**Kubernetes Replication controller:**

Which responsible for managing the pod lifecycle.

To scale up and scale down the pod replicas

Manifest file (replica=4)

To get the replicas:

* kubectl get rc

rc.yml:

apiVersion: v1

kind: ReplicationController

metadata:

name: rctomcat

spec:

replicas: 1

selector:

app: mytomcatserver

env: preprodserver

template:

metadata:

name: tomcat

labels:

app: mytomcatserver

env: preprodserver

spec:

containers:

- name: rctomcat

image: middlewaredevops4/tomcatimagewithapp

ports:

- containerPort: 8080

----------------------------

kubectl get rc

kubectl get pods

kubectl scale replicationcontroller <rcname> --replicas= <number>

kubectl scale replicationcontroller rctomcat --replicas=2

kubectl delete pod <podname>

kubectl delete pod rctomcat-hlkwq

To get the ip of pod:

> kubectl get pods -o wide

To access the application: <http://10.244.1.3:8080/supermarket/> (Login to respective node and use this url to access application)

<https://github.com/devopsmiddleware1/klabs>

* kubectl create -f <https://raw.githubusercontent.com/devopsmiddleware1/klabs/master/rc.yml>

<https://github.com/devopsmiddleware1/klabs.git>

Namespaces: provide for a scope of Kubernetes objects. You can think of it as a workspace you are sharing with other users.

namespace example

kubectl get ns

kubectl create namespace flipkartproject

kubectl get ns

kubectl run flipkartpod --image=middlewaredevops4/tomcatimagewithapp --port=8080 -n flipkartproject

kubectl get pods

kubectl get pods --namespace flipkartproject

nc.yml:

apiVersion: v1

kind: Namespace

metadata:

name: amazonproject

port forward

kubectl port-forward flipkartpod 9990:8080 --namespace flipkartproject

**Service Type:**

A service can be defined as logical set of pods. Abstraction on the top of the pod which provides a single IP ADDRESS AND dns name by which pods can be accessed.

<https://github.com/devopsmiddleware1/klabs/blob/master/tomcatsvc.yml>

<https://github.com/devopsmiddleware1/klabs/blob/master/rc.yaml>

Steps to Setup the dashboard\_new

Step-1: Deploy the dashboard Application

kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml (dont use this)

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml

Step-2: create user for Dashboard using below yaml file

ls

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

kubectl create -f usercreation.yml

Step-3: create a service account using below yaml file

vi serviceaccount.yml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

kubectl create -f serviceaccount.yml

kubectl get serviceaccount --all-namespaces | grep dashboard

step-4 ----- capturing the token for admin-user

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

copy the token

like this:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImFoVjZZQmM0RFM2SjVFR2RrV1NDWVE3YjgtQ2VDZmhFdVpmLWM5QmcwalEifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLWw3bmo0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJmODYxYTgxZi1lYzk4LTQwZWItYWM1NC0yMmM4YjQ0YjYyMzgiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.eJHqg3uS\_wyYXIDjGWf3TLfyADBmX5s2NjTk-UiV9Deg6CJDRlKgaEIMQJYdcN769JW2RCH6116CAynEEfEckVYmHSRzMg31nc8HmWUur1Gvat-GWcrbP8jsdJm8tKFcpICoj8bbtQu4azBMXdnjaT9L\_uSWD\_4YhiuUgArttv\_ve8gIKidp2Z-XI-\_j1jcOpUWlZ\_3iPDOMT5XXRpD3o7v1WOynUaZsq6kil3PxRi\_lYXHuyualMLgYauzoUA3QowA2pNBrvKq-1W7y5T8S-WzU8j\_TpgtE8e5vi21iG8PNor8ihbMG6PqTBBLfhAQjLh0HKjLxQHu4bWCfpupz3Q

Copy this

Step-5 access the console Kubectl will make Dashboard available at below url

kubectl proxy

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/. (not working)

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/. (latest 2021 working)

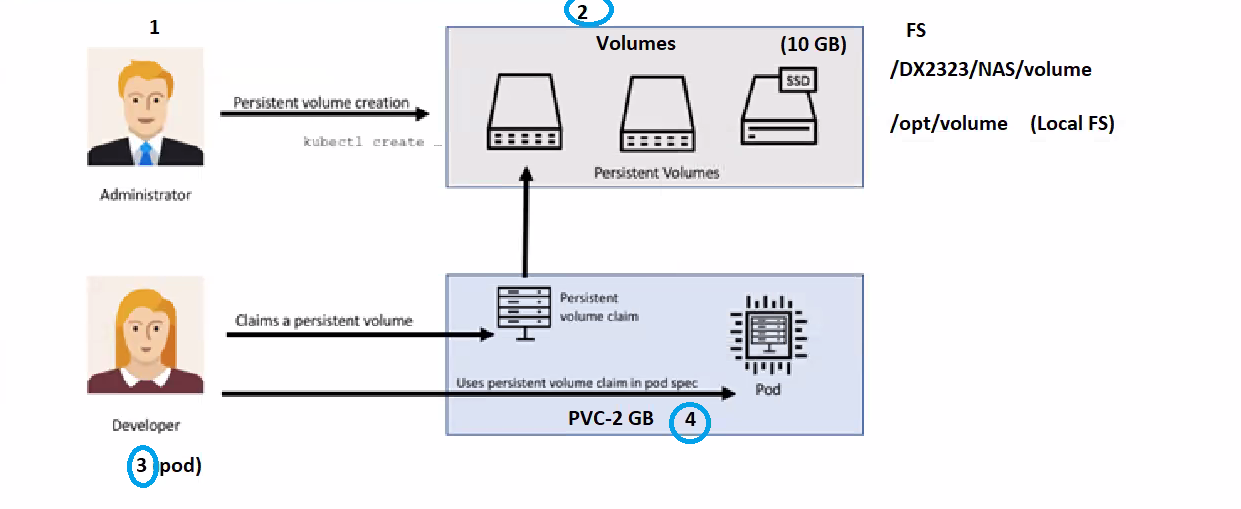
192.168.87.141

http://192.168.87.141:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/.

Token

paste the token

Kubernetes Volume:



Essentially a directory accessible to all containers running in a pod.

To check the persistence volume:

>kubctl get pv

To check the persistence volume claim:

>kubctl get pvc

Create volumes:

<https://github.com/devopsmiddleware1/klabs/tree/master/volume>

after executing above 3 files, volume will be created.

Go to volume and add some html file to verify volume functionality by access it through ip of the prod.

Get pods with ips:

* kubectl get pods -o wide

Deployments:

Deployment controller provides declarative updates for pods & replicasets.

Environment varaiables:

You can set environment while ceraeting containers of pod:

env or envFrom

To delete replica containers:

* kubectl delete rc rctomcat

To get replicat sets:

* kubectl get rc

>kubectl rollout status deploy/tomcat

>kubectl rollout history deploy/tomcat

>kubectl rollout undo deploy/tomcat –to-revision=1

>kubectl rollout history deploy/tomcat

>kubetcl get rc

<https://github.com/devopsmiddleware1/klabs/tree/master/deployment>

Deployment commands

kubectl create -f deployment1.yml

kubectl get deploy

kubectl get rs

kubectl get pods

kubectl apply -f deployment2.yml

kubectl rollout status deploy/tomcat

kubectl rollout history deploy/tomcat

kubectl rollout undo deploy/tomcat --to-revision=1

kubectl get deploy

kubectl get rs

kubectl get pods

Autoscaling:

Horizontal scalling

kubectl autoscale deployment <deployment-name> --min=0 --max=0 --cpu-percent=00

kubectl autoscale deployment tomcat --min=2 --max=4 --cpu-percent=80

kubectl get hpa

Docker Compose:

Kcompose = kubernates +docker compose

<https://github.com/devopsmiddleware1/klabs/tree/master/kompose>

To ckeck compse

* docker kcompose

To install docker compose

>yum install -y epel-release

>yum install -y kcompose

yum install -y epel-release

yum install -y kompose

kompose version

kompose convert -f docker-compose.yml -o komposeapp1.yml

kompose up

kompose down

**Secrets:**

Can be defined as Kubernetes objects used to store sensitive data such username and pwd with encryption.

Secrets are namespaced objects.

To delete deployment:

>kubectl delete deploy tomcat

>kubectl describe secret tom-secrets

<https://github.com/devopsmiddleware1/klabs/tree/master/secrets>

secret creation using cli

kubectl create secret generic tom2-secrets --from-file=username=uname.txt --from-file=password=pass.txt

kubectl describe secret tom2-secrets

kubectl get secrets

kubectl create -f podsec\_test.yml

--------------------------

secret creation using yaml file

echo -n 'admin2' | base64

YWRtaW4y

echo -n 'test1234$' | base64

dGVzdDEyMzQk

kubectl create -f secretcreation.yml

kubectl describe secret tom2-secrets

kubectl get secrets

kubectl create -f podsec\_test.yml

**ConfigMaps:**

Api resource holds key-value paris of configuration data that can be consumed in pods or used to store configuration data for system components such as a controllers

To get configmap

>kubectl get cm

Kubectl exe -it tomcatpodcm

Configmap: <https://github.com/devopsmiddleware1/klabs/tree/master/configmap>

kubectl get cm

kubectl create -f tomcatcm.yml

kubectl create -f tome.yml

to test the config map, get into pod check for env

kubectl exec -it <podname> /bin/bash

env

kubectl create -f <https://github.com/devopsmiddleware1/klabs/blob/master/2container.json>

getting into the pod

kubectl exec -it <podname> sh

kubectl exec -it tomcat-nginx sh

kubectl exec -it <podname> -c <containername> sh

kubectl exec -it tomcat-nginx -c nginx sh

>kubectl exec -it tomcat-nginx tomcat sh

Ingress:

<https://github.com/devopsmiddleware1/kubernetes_ingress_example>