KUBERNETES :

Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. provides a **container-centric** management environment. It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of your scaling requirements, failover, deployment patterns, and more.

**Traditional deployment era:** Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.

**Virtualized deployment era:** As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server’s CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware

**Container deployment era:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight.

FEATURES;

1. a container platform
2. a microservices platform
3. a portable cloud platform and a lot more.
4. Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
5. allows you to automatically mount a storage system of your choice,
6. you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
7. allows you to specify how much CPU and memory (RAM) each container needs.
8. restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to clients until they are ready to serve.
9. store and manage sensitive information
10. deployment, scaling, load balancing, logging, and monitoring.
11. platform for building an ecosystem of components and tools to make it easier to deploy, scale, and manage applications.

why containers :

* old way : applications on host , heavy , non portable , relies on package manager
* new way : deploy containers (app+libraries) , small , fasr , os level virtualization

components :

* master components
* node components
* addons

**master components :**

Master components make global decisions about the cluster (for example, scheduling), and detecting and responding to cluster events

Components :

**Kube apiserver** : Component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.It is designed to scale horizontally – that is, it scales by deploying more instances.

**Etcd**: Kubernetes’ backing store for all cluster data.Always have a backup plan for etcd’s data for your Kubernetes cluster.

**Kube-scheduler** :: watches newly created pods that have no node assigned, and selects a node for them to run on. Factors for scheduling : collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines

**Kube –controller-manager**: run controllers (nde , replication , endpoints , service)

* Node Controller: Responsible for noticing and responding when nodes go down.
* Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
* Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
* Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces

**Cloud-controller-manager :** runs controllers that interact with the underlying cloud providers. Disable by –cloud-provider=external.

launch a single node cluster

* minikube version
* Start the cluster, by running the minikube start command: minikube start

Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

* **Kubectl**. Kubectl uses the Kubernetes API to interact with the cluster. When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run.

**Example** : Node.js application packaged in a Docker container. To create the Node.js application and deploy the Docker container,

* cluster and its health status can be discovered via : kubectl cluster info
* view the nodes in the cluster using : kubectl get nodes
* With a running Kubernetes cluster, containers can now be deployed.

Using kubectl run, it allows containers to be deployed onto the cluster - kubectl run first-deployment --image=katacoda/docker-http-server --port=80

* The status of the deployment can be discovered via the running Pods - kubectl get pods
* Once the container is running it can be exposed via different networking options, depending on requirements. One possible solution is NodePort, that provides a dynamic port to a container.
* kubectl expose deployment first-deployment --port=80 --type=NodePort
* The command below finds the allocated port and executes a HTTP request.
* export PORT=$(kubectl get svc first-deployment -o go-template='{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}') echo "Accessing host01:$PORT" curl host01:$PORT
* Enable the dashboard using Minikube with the command minikube addons enable dashboard
* Make the Kubernetes Dashboard available by deploying the following YAML definition. This should only be used on Katacoda.
* kubectl apply -f /opt/kubernetes-dashboard.yaml
* The Kubernetes dashboard allows you to view your applications in a UI. In this deployment, the dashboard has been made available on port *30000*.

launch a multinode cluster using kubeadm

Kubeadm solves the problem of handling TLS encryption configuration, deploying the core Kubernetes components and ensuring that additional nodes can easily join the cluster. The resulting cluster is secured out of the box via mechanisms such as RBAC.

Server.js

**var http = require('http');**

**var handleRequest = function(request, response) {**

**console.log('Received request for URL: ' + request.url);**

**response.writeHead(200);**

**response.end('Hello World!');**

**};**

**var www = http.createServer(handleRequest);**

[**www.listen(8080)**](http://www.listen(8080))**;**

**dockerfile**

**FROM node:6.14.2**

**EXPOSE 8080**

**COPY server.js .**

**CMD node server.js**

**Docker build .**

**Docker create <image>**

**Docker ps –a**

**Docker images**

**Minikube install : curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \**

**&& chmod +x minikube**

**sudo install minikube /usr/local/bin**

kubernetes :

Kubernetes can be installed and deployed using following methods:

* Minikube ( It is a single node kubernetes cluster)
* Kops ( Multi node kubernetes setup into AWS )
* Kubeadm ( Multi Node Cluster in our own premises)

On the Master Node following components will be installed

* **API Server**  – It provides kubernetes API using Jason / Yaml over http, states of API objects are stored in etcd
* **Scheduler**– It is a program on master node which performs the scheduling tasks like launching containers in worker nodes based on resource availability
* **Controller Manager** – Main Job of Controller manager is to monitor replication controllers and create pods to maintain desired state.
* **etcd** – It is a Key value pair data base. It stores configuration data of cluster and cluster state.
* **Kubectl utility** – It is a command line utility which connects to API Server on port 6443. It is used by administrators to create pods, services etc.

On Worker Nodes following components will be installed

* **Kubelet** – It is an agent which runs on every worker node, it connects to docker  and takes care of creating, starting, deleting containers.
* **Kube-Proxy** – It routes the traffic to appropriate containers based on ip address and port number of the incoming request. In other words we can say it is used for port translation.
* **Pod** – Pod can be defined as a multi-tier or group of containers that are deployed on a single worker node or docker host.

Docker commands :

# Check Docker hub for images with a given name

docker images ( list all images )

docker search ubuntu (search ubuntu in all images )

# Pull a container from Docker hub. To use an image tag - use [image-name]:[tag]

docker pull hello-world

docker pull ubuntu

docker pull ubuntu:12.10

# List all running containers

docker ps (active)

docker ps -a (active / not active)

# List what is running in a container

docker top 32c5a521c664

# Start/run a container - pull it from Docker hub if not present. You can name it.Standard output/error will be redirected to the current terminal

docker run hello-world

docker run --name=test-container-name hello-world

# Start in detached mode - standard output/error will NOT be redirected to the current terminal

docker run -d hello-world

# Start a container and run a specific command from within

docker run busybox echo "Test\n"

docker run ubuntu /bin/echo hello ubuntu container

run ubuntu /bin/bash -c "ls -l"

# Start a container and run a bash sessions interactively (SSH-style)

docker run -it ubuntu bash

docker run -it ubuntu /bin/bash

docker run -it busybox sh

# Attach a shell into a running container (SSH-style). Id/Name is taken from ps.

sudo docker exec -it b3a04a93f46f bash

sudo docker exec -it b3a04a93f46f sh

# Detach from a container's shell without killing it (keyboard shortcut):

Ctrl + p + q

# Reattach/reconnect to running shell (if doconnected with the above shortcut)

docker attach 32c5a521c664

# Run an arbitrary command within a running container

docker exec b3a04a93f46f /bin/echo "Hello again"

# Start a web server in a container; map its port (80) to a localhost port (8080)

docker run -p 8080:80 nginx

# If containers can't access the hosts networking (run on host)

sysctl -w net.ipv4.ip\_forward=1

# Force the container to use a specific DNS server

docker run --dns 8.8.8.8 ubuntu

# Mount a host folder '~/test-vol' to a container as '/host-test-vol'

docker run -it -v ~/test-vol:/host-test-vol ubuntu /bin/bash

# Start/restart a stopped/running container

docker start 33712928d6c5

docker restart 33712928d6c5

# Gracefully stop a running container by.

# Sends SIGTERM to the root process.

docker stop b3a04a93f46f

# Forcefully stop a running container (SIGKILL)

docker kill b3a04a93f46f

# Remove a stopped container

docker rm b3a04a93f46f

# Remove a container regardless of its state (i.e. running, stopped)

docker rm -f b3a04a93f46f

# Check the changes in a container, since it was started from an image

docker diff 60b4f89dfc7f

# Create an image from a container - it will include all new changes

docker commit 60b4f89dfc7f test-image-name

issue :

sock error :

sudo chmod 777 /var/run/docker.sock

kubernetes commands

## Pull the image from the Repository and create a Container on the Cluster

kubectl run my-app --image=gcr.io/some-repo/my-app:v1 –port=3000

Kubernetes groups containers together in ‘Pods’

kubectl get pods

## Expose the Kubernetes Deployment through a Load Balancer

kubectl expose deployment my-app --type=LoadBalancer --port=8080 --target-port=3000

## Find the external IP of your Container

kubectl get svc

## Use Kubernetes Rolling Updates

K8s to update our application with just one command

kubectl set image deployment/my-app my-app=gcr.io/some-repo/my-app:v2

clean up :

$ kubectl delete deployment my-app

$ kubectl delete svc my-app

Kubernetes cluster orchestration system . Containerization helps package software to serve these goals, enabling applications to be released and updated in an easy and fast way without downtime. Kubernetes helps you make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work.

* Create cluster
* deploy app
* explore ur app
* expose publically
* scale ur app
* update ur app

pre-requisites :

1. curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
2. chmod +x ./kubectl
3. sudo mv ./kubectl /usr/local/bin/kubectl
4. kubectl version
5. systemctl daemon-reload
6. systemctl restart kubelet

* apt-get update && apt-get install -y apt-transport-https curl
* curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
* cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
* deb https://apt.kubernetes.io/ kubernetes-xenial main
* EOF
* apt-get update
* apt-get install -y kubelet kubeadm kubectl

apt mark hold kubelet kubeadm kubectl

create cluster :

apt-get update && apt-get upgrade or yum update

sudo swapoff -a

sudo apt-get install kubeadm -y

kubeadm version

initialize network

sudo kubeadm init --pod-network-cidr=172.31.4.0/20

mkdir -p $HOME/.kube

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

sudo chmod +x /home/impadmin/.kube/config

check nodes

kubectl get nodes

install pod network

kubectl apply -f <add-on.yaml>

kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/62e44c867a2846fefb68bd5f178daf4da3095ccb/Documentation/kube-flannel.yml>

kubectl get pods --all-namespaces

joining nodes :

kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>

kubeadm token list

kubeadm token create

drain the node ;

kubectl drain <node name> --delete-local-data --force –ignore-daemonsets

kubectl delete node <node name>

kubeadm reset

iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X

kubectl :

kubectl [command] [TYPE] [NAME] [flags]

command : create , delete , describe , get

flag : -f file1 -f file2 -f file

kubectl get pods

kubectl describe pods 1-75fdfdc8c5-dwmdj

kubectl logs **${**POD\_NAME**}** **${**CONTAINER\_NAME**}**

kubectl delete pods 1-75fdfdc8c5-h7c8g --grace-period=0 –force

script to fetch name , status , iamge id from kubernetes describe command :

#!/bin/bash

pods=`kubectl get pods |awk '{print $1}'|awk '{if(NR>1)print}'| xargs`

echo "pods and there status are: $name"

name= kubectl describe pod $pods > pods.txt

grep -e "^\\Name" -e "^\\Status" -e Image -e "^\\Age" pods.txt

KUBERNETES CLUSTER SETUP USING ANSIBLE :

Dependencies.yml

---

- name: install Docker

yum:

name: docker

state: present

update\_cache: true

- name: start Docker

service:

name: docker

state: started

- name: disable SELinux

command: setenforce 0

ignore\_errors: yes

- name: disable SELinux on reboot

selinux:

state: disabled

ignore\_errors: yes

- name: ensure net.bridge.bridge-nf-call-ip6tables is set to 1

sysctl:

name: net.bridge.bridge-nf-call-ip6tables

value: 1

state: present

- name: ensure net.bridge.bridge-nf-call-iptables is set to 1

sysctl:

name: net.bridge.bridge-nf-call-iptables

value: 1

state: present

- name: add Kubernetes YUM repository

yum\_repository:

name: Kubernetes

description: Kubernetes YUM repository

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

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

gpgcheck: yes

- name: install kubelet

yum:

name: kubelet-1.14.0

state: present

update\_cache: true

- name: install kubeadm

yum:

name: kubeadm-1.14.0

state: present

- name: start kubelet

service:

name: kubelet

enabled: yes

state: started

- name: install kubectl

yum:

name: kubectl-1.14.0

state: present

allow\_downgrade: yes

hosts

[masters]

master ansible\_host=172.26.41.67 ansible\_user=root

[workers]

worker1 ansible\_host=172.26.41.87 ansible\_user=root

master.yml

---

- name: Clean up befor init Cluster

shell: yes | kubeadm reset

- name: clean existing network

shell: iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X

- name: initialize the cluster

shell: kubeadm init --pod-network-cidr=10.244.0.0/16

- name: create .kube directory

file:

path: /root/.kube

state: directory

mode: 0755

- name: copy admin.conf to user's kube config

copy:

src: /etc/kubernetes/admin.conf

dest: /root/.kube/config

remote\_src: yes

- name: install Pod network

shell: kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878db11b/Documentation/kube-flannel.yml>

get\_join.yml

---

- name: wait for cluster up

wait\_for:

port: "{{item}}"

delay: 120

with\_items:

- 6443

- name: get join command

shell: kubeadm token create --print-join-command

register: join\_command\_raw

- name: set join command

set\_fact:

join\_command: "{{ join\_command\_raw.stdout\_lines[0] }}"

join.yml

---

- name: Clean up befor joining Cluster

shell: yes | kubeadm reset

- name: clean existing network

shell: iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X

- name: join cluster

shell: "{{ hostvars['master'].join\_command }} --ignore-preflight-errors all"

vars.yml

common\_dep: true ----- dependencies.yml

ini\_cluster: true ---- master.yml

join\_cluster: true -- get\_join.yml and join.yml

install\_dashboard: true ------- dashboard.yml

test.yml

- hosts: all

become: yes

tasks:

- name: Bare include (free-form)

include\_vars: vars.yml

- include\_tasks: dependencies.yml

when: common\_dep

- hosts: master

become: yes

tasks:

- include\_tasks: master.yml

when: ini\_cluster

- hosts: master

become: yes

tasks:

- include\_tasks: get\_join.yml

when: join\_cluster

- hosts: workers

become: yes

tasks:

- include\_tasks: join.yml

when: join\_cluster

- hosts: master

become: yes

tasks:

- include\_tasks: dashboard.yml

when: install\_dashboard

dashboard.yml

---

- name: install K8s Dashboard

shell: kubectl apply -f /root/devops/Kubernetes/Kubernetes\_dashboard.yml

Kubernetes\_dashboard.yml

---

- name: install K8s Dashboard

shell: kubectl apply -f /root/devops/Kubernetes/Kubernetes\_dashboard.yml

[root@IMPETUS-I0065 test]# cat /root/devops/Kubernetes/Kubernetes\_dashboard.yml

# Copyright 2017 The Kubernetes Authors.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

# ------------------- Dashboard Secrets ------------------- #

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-certs

namespace: kube-system

type: Opaque

---

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-csrf

namespace: kube-system

type: Opaque

data:

csrf: ""

---

# ------------------- Dashboard Service Account ------------------- #

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

---

# ------------------- Dashboard Role & Role Binding ------------------- #

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

rules:

# Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.

- apiGroups: [""]

resources: ["secrets"]

verbs: ["create"]

# Allow Dashboard to create 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["create"]

# Allow Dashboard to get, update and delete Dashboard exclusive secrets.

- apiGroups: [""]

resources: ["secrets"]

resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs", "kubernetes-dashboard-csrf"]

verbs: ["get", "update", "delete"]

# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

# Allow Dashboard to get metrics from heapster.

- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:"]

verbs: ["get"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: kubernetes-dashboard-minimal

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kube-system

---

# ------------------- Dashboard Deployment ------------------- #

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

spec:

containers:

- name: kubernetes-dashboard

image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1

ports:

- containerPort: 8443

protocol: TCP

args:

- --auto-generate-certificates

# Uncomment the following line to manually specify Kubernetes API server Host

# If not specified, Dashboard will attempt to auto discover the API server and connect

# to it. Uncomment only if the default does not work.

# - --apiserver-host=http://my-address:port

volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs

# Create on-disk volume to store exec logs

- mountPath: /tmp

name: tmp-volume

livenessProbe:

httpGet:

scheme: HTTPS

path: /

port: 8443

initialDelaySeconds: 30

timeoutSeconds: 30

volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs

- name: tmp-volume

emptyDir: {}

serviceAccountName: kubernetes-dashboard

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

---

# ------------------- Dashboard Service ------------------- #

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

ports:

- port: 443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboard

type: NodePort

COMMANDS :

kubectl get pod -n kube-system

kubectl get svc -n kube-system

kubectl get svc

kubectl get node

README KUBERNETES CLUSTER SETUP USING ANSIBLE :

1. Move to location :
   1. /root/kubernetes/test
2. Yml and their roles :
   1. **dependencies.yml** (optional): to install docker , disable selinux . install kublet , install kubeadm , install kubectl , and to start them if not done earlier
   2. **hosts :** add all the nodes , where cluster needs to be setup
   3. **cluster\_initialization.yml :** cleanup before initialization and install pod network
   4. **create\_token.yml :** create a bootstrap token , Instead of printing only the token, print the full 'kubeadm join' flag needed to join the cluster using the token.
   5. **joining\_cluster.yml :** clean existing cluster and join the cluster .
   6. **vars.yml :** contains variables based on condition
   7. **main.yml :** The main playbook which will handle execution , based on the variable input (true/false) in vars.yml
   8. **configure\_dashboard.yml :** create and configure dashboard
3. Add the required nodes in the hosts file

syntax :

[masters]

master ansible\_host=172.26.41.79 ansible\_user=root ansible\_ssh\_pass=impetus@123

[workers]

worker1 ansible\_host=172.26.41.26 ansible\_user=root ansible\_ssh\_pass=impetus@123

1. Edit vars.yml , depending on the requirement

Set variables ro true/false

1. After making changes to hosts file and to the vars file , run the playbooks with the following :

ansible-playbook -i hosts main.yml -vvv