**Kubernatis (k8s) Basics**

Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers.

With Kubernetes,

Can quickly and efficiently respond to customer demand:

Deploy applications quickly and predictably.

Scale applications on the fly.

Roll out new features seamlessly.

Limit hardware usage to required resources only.

Goal is to foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds.

Kubernetes is

* Open source container cluster manager
* Developed by Google
* Used in google cloud
* Released in July 2015
* Goal is to automate deployment, scaling, maintain high availability
* Other Cluster management/Orchestration Tools:

Docker Swarm

Apache Mesos

* Owned by Seed Technology now

**Components:**

Nodes – minions

Pods

Labels

Selectors

Controllers

Services

Control Pane

API

**Architecture:**

Nodes can be virtual or physical hosts

Docker has to be installed on all of the nodes

Each minion will run ETCD (key pair management and communication service ) for exchange of messages and reporting on cluster status

**Pods**

Pod consists of one or more containers, these containers are located on the same host.

They share the resources

Pods are assigned a unique IPs with in cluster

Pod management is done through the API

**Labels:**

Clients can attach key-value pairs to any object in the system

Grouping is done using labels and selectors

**Controllers:**

For management of cluster

Manage set of Pods depending on the desired state of the cluster

Service:

set of pods can work together , they can defined and implement a service ex: mysql or Apache

**Installation and configuration**

1. Install ntp on all the servers

***yum install –y ntp***

1. Start the service on all nodes

***systemctl enable ntpd && systemctl start ntpd***

1. Add ipaddress of each machine in /etc/hosts of all machines

[root@master ~]# cat /etc/hosts

10.0.0.81 master

10.0.0.144 minion1

10.0.0.61 minion2

1. Update file as below

***vi /etc/yum.repos.d/virt7-docker-common-release.repo***

[virt7-docker-common-release]

name=virt7-docker-common-release

baseurl=http://cbs.centos.org/repos/virt7-docker-common-release/x86\_64/os/

gpgcheck=0

1. Enable the repo

***yum install –enablerepo=virt7-docker-common-release etcd kubernetes docker –y***

**Configuring Master:**

=================

Add the information about the master controller

Configure 1 node as master

Configure 2 nodes as minions minion1 and minion2

1. Change the configuration file

***vi /etc/kubernetes/config***

Change the KUBE\_MASTER as below

MASTER=”--http://master:8080”

KUBE\_ETCD\_SERVERS=”—etcd-servers=http://master:2379”

1. Changed the etcd configuration

***vi /etc/etcd/etcd.conf***

Change below:

In the [member] section

ETCD\_LISTEN\_CLIENT\_URLS=<http://0.0.0.0:2379>

In the cluster section

ETCD\_ADVERTISE\_CLIENT\_URLS=<http://0.0.0.0:2379>

1. Edit the API server

***vi /etc/kubernets/apiserver***

KUBE\_API\_ADDRESS=”—address=0.0.0.0”

Uncomment

KUBR\_API\_SEVER=”—port=8080”

KUBELET\_PORT=”—kubelet-port=10250”

KUBE\_SERVICE\_ADDRESSES=”—service-cluster-ip-range=10.254.0.0/16”

On master

***systemctl enable etcd kube-apiserver kube-controller-manager kube-scheduler***

***systemctl start etcd kube-apiserver kube-controller-manager kube-scheduler***

**Configuring the minions:**

====================

1. Change the configuration file

***vi /etc/kubernetes/config***

Change the KUBE\_MASTER as below

MASTER=”--http://master:8080”

KUBE\_ETCD\_SERVERS=”—etcd-servers=http://master:2379”

1. Change the kubelet configuration

***vi /etc/kubernetes/kubelet***

KUBELET\_ADDRESS=”—address=0.0.0.0”

Uncomment kubelet port

KUBELET\_PORT=”—port=10250”

KUBELET\_HOSTNAME=”—hostname-override=minion1”

KUBELET\_API\_SERVER=”—api-servers=http://master:8080”

Comment out KUBELET\_POD\_INFRA\_CONTAINER

1. Start the kube services on all minions

***systemctl enable kube-proxy kubelet docker***

***systemctl start kube-proxy kubelet docker***

Perform same steps on all minions.

with this the configuration is complete

On master

**[root@master ~]# kubectl get nodes**

NAME STATUS AGE

minion1 Ready 4m

minion2 Ready 3m

[root@master ~]#

[root@master ~]# kubectl describe nodes

Name: minion1

Role:

Labels: beta.kubernetes.io/arch=amd64

beta.kubernetes.io/os=linux

kubernetes.io/hostname=minion1

Taints: <none>

CreationTimestamp: Sun, 15 Oct 2017 03:32:08 +0000

Phase:

Conditions:

Type Status LastHeartbeatTime LastTransitionTime Re

ason Message

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

---- -------

OutOfDisk False Sun, 15 Oct 2017 03:37:39 +0000 Sun, 15 Oct 2017 03:32:08 +0000 Ku

beletHasSufficientDisk kubelet has sufficient disk space available

MemoryPressure False Sun, 15 Oct 2017 03:37:39 +0000 Sun, 15 Oct 2017 03:32:08 +0000 Ku

beletHasSufficientMemory kubelet has sufficient memory available

DiskPressure False Sun, 15 Oct 2017 03:37:39 +0000 Sun, 15 Oct 2017 03:32:08 +0000 Ku

beletHasNoDiskPressure kubelet has no disk pressure

Ready True Sun, 15 Oct 2017 03:37:39 +0000 Sun, 15 Oct 2017 03:32:18 +0000 Ku

beletReady kubelet is posting ready status

Addresses: 10.0.0.144,10.0.0.144,minion1

Capacity:

alpha.kubernetes.io/nvidia-gpu: 0

cpu: 2

memory: 500248Ki

pods: 110

Allocatable:

alpha.kubernetes.io/nvidia-gpu: 0

cpu: 2

memory: 500248Ki

pods: 110

System Info:

Machine ID: 980d6e7da5004216ade783778573fd3b

System UUID: D5F63352-C6CD-4EEF-BB73-90368BD7677E

Boot ID: ccb89bc5-6da5-430c-8da0-40899d370d84

Kernel Version: 3.10.0-514.26.2.el7.x86\_64

OS Image: CentOS Linux 7 (Core)

Operating System: linux

Architecture: amd64

Container Runtime Version: docker://1.12.6

Kubelet Version: v1.5.2

Kube-Proxy Version: v1.5.2

ExternalID: minion1

Non-terminated Pods: (0 in total)

Namespace Name CPU Requests CPU Limits Memory Requests Memory Limits

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

Allocated resources:

(Total limits may be over 100 percent, i.e., overcommitted.

CPU Requests CPU Limits Memory Requests Memory Limits

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

0 (0%) 0 (0%) 0 (0%) 0 (0%)

Events:

FirstSeen LastSeen Count From SubObjectPath Type Reason M

essage

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

------

11m 11m 1 {kube-proxy minion1} Normal Starting S

tarting kube-proxy.

5m 5m 1 {kubelet minion1} Normal Starting S

tarting kubelet.

5m 5m 1 {kubelet minion1} Warning ImageGCFailed u

nable to find data for container /

5m 5m 2 {kubelet minion1} Normal NodeHasSufficientD

isk Node minion1 status is now: NodeHasSufficientDisk

5m 5m 2 {kubelet minion1} Normal NodeHasSufficientM

emory Node minion1 status is now: NodeHasSufficientMemory

5m 5m 2 {kubelet minion1} Normal NodeHasNoDiskPress

ure Node minion1 status is now: NodeHasNoDiskPressure

5m 5m 1 {kubelet minion1} Normal NodeReady N

ode minion1 status is now: NodeReady

Name: minion2

Role:

Labels: beta.kubernetes.io/arch=amd64

beta.kubernetes.io/os=linux

kubernetes.io/hostname=minion2

Taints: <none>

CreationTimestamp: Sun, 15 Oct 2017 03:33:10 +0000

Phase:

Conditions:

Type Status LastHeartbeatTime LastTransitionTime Re

ason Message

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

---- -------

OutOfDisk False Sun, 15 Oct 2017 03:37:41 +0000 Sun, 15 Oct 2017 03:33:10 +0000 Ku

beletHasSufficientDisk kubelet has sufficient disk space available

MemoryPressure False Sun, 15 Oct 2017 03:37:41 +0000 Sun, 15 Oct 2017 03:33:10 +0000 Ku

beletHasSufficientMemory kubelet has sufficient memory available

DiskPressure False Sun, 15 Oct 2017 03:37:41 +0000 Sun, 15 Oct 2017 03:33:10 +0000 Ku

beletHasNoDiskPressure kubelet has no disk pressure

Ready True Sun, 15 Oct 2017 03:37:41 +0000 Sun, 15 Oct 2017 03:33:20 +0000 Ku

beletReady kubelet is posting ready status

Addresses: 10.0.0.61,10.0.0.61,minion2

Capacity:

alpha.kubernetes.io/nvidia-gpu: 0

cpu: 2

memory: 500248Ki

pods: 110

Allocatable:

alpha.kubernetes.io/nvidia-gpu: 0

cpu: 2

memory: 500248Ki

pods: 110

System Info:

Machine ID: 980d6e7da5004216ade783778573fd3b

System UUID: D5CE9FAF-C19F-43AF-A2B8-8D2D07799368

Boot ID: 7f676c28-3494-472c-a900-e57bad33f042

Kernel Version: 3.10.0-514.26.2.el7.x86\_64

OS Image: CentOS Linux 7 (Core)

Operating System: linux

Architecture: amd64

Container Runtime Version: docker://1.12.6

Kubelet Version: v1.5.2

Kube-Proxy Version: v1.5.2

ExternalID: minion2

Non-terminated Pods: (0 in total)

Namespace Name CPU Requests CPU Limits Memory Requests Memory Limits

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

Allocated resources:

(Total limits may be over 100 percent, i.e., overcommitted.

CPU Requests CPU Limits Memory Requests Memory Limits

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

0 (0%) 0 (0%) 0 (0%) 0 (0%)

Events:

FirstSeen LastSeen Count From SubObjectPath Type Reason M

essage

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

------

4m 4m 1 {kube-proxy minion2} Normal Starting S

tarting kube-proxy.

4m 4m 1 {kubelet minion2} Normal Starting S

tarting kubelet.

4m 4m 1 {kubelet minion2} Warning ImageGCFailed u

nable to find data for container /

4m 4m 2 {kubelet minion2} Normal NodeHasSufficientD

isk Node minion2 status is now: NodeHasSufficientDisk

4m 4m 2 {kubelet minion2} Normal NodeHasSufficientM

emory Node minion2 status is now: NodeHasSufficientMemory

4m 4m 2 {kubelet minion2} Normal NodeHasNoDiskPress

ure Node minion2 status is now: NodeHasNoDiskPressure

4m 4m 1 {kubelet minion2} Normal NodeReady N

ode minion2 status is now: NodeReady

**To get information about nodes:**

***kubectl get nodes***

**Creating POD using yaml file:**

[root@master test]# cat nginx.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

To create POD:

kubectl create -f ./nginx.yaml

To list PODs:

Kubectl get pods

Creation of POD may fail because of a bug

**Work around for bug in kubernetes:**

=================================

https://github.com/kubernetes/kubernetes/issues/11355#issuecomment-127378691

Restart all the process once a key is generated

Label is to identity the POD in thousands of PODS , it is like a tag

It is a key value pair

[root@master test]# cat nginx1.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx1

labels:

app: nginx1

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

***kubectl create –f nginx1.yaml***

* Creates pod nginx1

***kubectl get-pods –l app=nginx***

* Description of pods

[root@master test]# kubectl get pods -l app=nginx

NAME READY STATUS RESTARTS AGE

nginx3 1/1 Running 0 18s

[root@master test]# kubectl describe pods -l app=nginx

Name: nginx3

Namespace: default

Node: minion2/10.0.0.40

Start Time: Sat, 21 Oct 2017 02:27:45 +0000

Labels: app=nginx

Status: Running

IP: 172.17.0.4

Controllers: <none>

Containers:

nginx1:

Container ID: docker://6be47b2532269229d7fcc6b1a878b15ed3dd1e4019e4894b8c375044c3934171

Image: nginx:1.7.9

Image ID: docker-pullable://docker.io/nginx@sha256:e3456c851a152494c3e4ff5fcc26f240206abac0c9d794aff

b40e0714846c451

Port: 80/TCP

State: Running

Started: Sat, 21 Oct 2017 02:27:48 +0000

Ready: True

Restart Count: 0

Volume Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-jglff (ro)

Environment Variables: <none>

Conditions:

Type Status

Initialized True

Ready True

PodScheduled True

Volumes:

default-token-jglff:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-jglff

QoS Class: BestEffort

Tolerations: <none>

Events:

FirstSeen LastSeen Count From SubObjectPath Type Reason M

essage

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

------

1m 1m 1 {default-scheduler } Normal ScheduledS

uccessfully assigned nginx3 to minion2

1m 1m 2 {kubelet minion2} Warning MissingClu

sterDNS kubelet does not have ClusterDNS IP configured and cannot create Pod using "ClusterFirst" policy. Falling

back to DNSDefault policy.

1m 1m 1 {kubelet minion2} spec.containers{nginx1} Normal Pulled C

ontainer image "nginx:1.7.9" already present on machine

1m 1m 1 {kubelet minion2} spec.containers{nginx1} Normal Created C

reated container with docker id 6be47b253226; Security:[seccomp=unconfined]

1m 1m 1 {kubelet minion2} spec.containers{nginx1} Normal Started S

tarted container with docker id 6be47b253226

**Deployments:**

**For production:**

For deployment, we need to use latest API which is in Extension

[root@master test]# cat nginx-deploy-production.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-deploy

spec:

replicas: 3

template:

metadata:

labels:

app: nginx-deploy

spec:

containers:

- name: nginx-deploy

image: nginx:1.7.9

ports:

- containerPort: 80

With deployment we can do rolling update to PODs,

Update nginx version from 1.7.9 to 1.8

***Kubectl apply nginx-deploy.yaml 🡪 read yaml file and update to 1.8***

Update production env to new level

**Kind:**

Pod 🡪 to create a POD definition

Deployment 🡪 For creating production like environment and perform update

Replicationcontroller 🡪 for high availability

Service 🡪 Acts as a load balancer to Kubernetes Cluster.

Kubernetes Pods are mortal. They are born and when they die, they are not resurrected. ReplicationControllers in particular create and destroy Pods dynamically (e.g. when scaling up or down or when doing rolling updates). While each Pod gets its own IP address, even those IP addresses cannot be relied upon to be stable over time.

This leads to a problem: if some set of Pods (let’s call them backends) provides functionality to other Pods (let’s call them frontends) inside the Kubernetes cluster,

**how do those frontends find out and keep track of which backends are in that set?**

Solution is services

A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

As an example, consider an image-processing backend which is running with 3 replicas. Those replicas are fungible - frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that or keep track of the list of backends themselves. The Service abstraction enables this decoupling.

For Kubernetes-native applications, Kubernetes offers a simple Endpoints API that is updated whenever the set of Pods in a Service changes. For non-native applications, Kubernetes offers a virtual-IP-based bridge to Services which redirects to the backend Pods.

**Documentation :**

https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport

**Examples:**

Example:

$ vi nginx\_pod.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: nginx

spec:

replicas: 2

selector:

app: nginx

template:

metadata:

name: nginx

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

apiVersion: v1

kind: Service

metadata:

name: api

namespace: wardle

spec:

ports:

- port: 443

protocol: TCP

targetPort: 443

selector:

apiserver: "true"

[root@minion1 ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS

PORTS NAMES

ae3fc2bb7b68 nginx "nginx -g 'daemon off" 35 minutes ago Up 35

minutes k8s\_nginx.a6022f15\_nginx-gr8ck\_default\_0535328e-b73e-11e7-8a52-080027ce8866\_7000

8e87

2d3378f028ca gcr.io/google\_containers/pause-amd64:3.0 "/pause" 35 minutes ago Up 35

minutes k8s\_POD.b2390301\_nginx-gr8ck\_default\_0535328e-b73e-11e7-8a52-080027ce8866\_0cb243

ea

[root@minion1 ~]#

A pause container is created by default on minon

Why?

In Kubernetes, each pod has an IP and within a pod there exists a so called infrastructure container, which is the first container that the Kubelet instantiates and it acquires the pod’s IP and sets up the network namespace. All the other containers in the pod then join the infra container’s network and IPC namespace. The infra container has network bridge mode enabled and all the other containers in the pod share its namespace via container mode. The initial process that runs in the infra container does effectively nothing since its sole purpose is to act as the home for the namespaces.

**Nginx Server Deployment using Kubernetes**

=======================================

1. Create pod yaml file

***vi nginx\_pod.yaml***

apiVersion: v1

kind: ReplicationController

metadata:

name: nginx

spec:

replicas: 2

selector:

app: nginx

template:

metadata:

name: nginx

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

1. Create pod

***kubectl create -f nginx\_pod.yaml***

1. Deploy the nginx service using yaml file in order to expose the nginx pod on the host port “82”

***$ vi nginx\_service.yaml***

apiVersion: v1

kind: Service

metadata:

labels:

name: nginxservice

name: nginxservice

spec:

ports:

# The port that this service should serve on.

- port: 82

# Label keys and values that must match in order to receive traffic for this service.

selector:

app: nginx

type: LoadBalancer

1. Create the nginx service using kubectl

***kubectl create -f nginx\_service.yaml***

services/nginxservice

The nginx service can be listed as follow

***kubectl get services***

NAME LABELS SELECTOR IP(S) PORT(S)

kubernetes component=apiserver,provider=kubernetes <none> 192.168.3.1 443/TCP

nginxservice name=nginxservice app=nginx 192.168.3.43 82/TCP