Kubernetes Overview

- Purpose of k8s is to host application in form of containers in automated way so you can deploy application and it can communicate with different services.

- The master node manage, plan, scheduler and monitor the nodes.

- The worker nodes hosts the application as containers.

Example:

1. Master is control ship which is responsible to manage the worker nodes which is having load.

2. Master node loads the container in the worker node as well as plan how to load, identify information of ship.

Master components

etcd cluster: It's database which stores the key value store in HA format i.e. Information of cluster

kube-scheduler: Identify right node to place container on based on container resource requirement. The worker node capacity or policy i.e Scheduling application

Controller-Manager:
1. Node-controller: Takes care of nodes i.e node availability.
2. Replication-controller: Take care of desired number of containers are up in replication group.

kube-apiserver: It's responsible for all operation in master node which control all service in master node. i.e orchestrating all operation within the cluster.

- Master node can be container as well

Docker should be install on all nodes (master/worker)

Worker Node Components

kubelet: Is caption of worker node (i.e agent)

kube-api server fetch status report from kubelet to monitor status of worker node and container running in them.

Engine:

kube-proxy:
- Communication between worker node is enable by kube-proxy.
- It ensures the essential rules in place so worker nodes to allow container to reach out to each other.

example: application and database container
##

ETCD

It is distributed reliable key value store which is simple, secure and fast.

what is key-value store:
- It is in tabular format (table)

Install ETCD

1. Download Binaries
curl -L https://github.com/etcd-io/etcd/releases/download/v3.3.11/etcd-v3.3.11-linux-amd64.tar.gz -o etcd-v3.3.11-linux-amd64.tar.gz

2. Extract
tar xzvf etcd-v3.3.11-linux-amd64.tar.gz

3. Run ETCD service
./etcd

- Run on 2379 port
- Can attach any client with ETCD service to store information for that client.
- Default client is ETCD control client, it is command line client for ETCD
- We can use ETCD control client to store and retrieve key-value pairs.

To store key value pair run:
./etcdctl set key1 <value1>
This will create entry in database with information

To retrieve key value pair:
./etcdctl get key1 ---> o/p = value1

To get more information:
./etcdctl
==========

ETCD Roles in k8s

- ETCD data store, stores the information about cluster.
i.e Nodes, PODs, Configs, Secrets, Accounts, Roles, Bindings, Others.

- When you run kubectl get command the information is collecting from ETCD server only.

- Every change in cluster ex: additional nodes, pods, replica sets are updated in ETCD server.

- If changes are updated in ETCD server then and only it consider as completed.
==========

Setup etcd

1. Manual
wget -q --https-only \
"https://github.com/coreos/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-am64.tar.gz"

2. Kubeadm

kubectl get pods -n kube-system

Run below command in etcd-master POD - It will explore ETCD for you.

kubectl exec etcd-master -n kube-system etcdctl get / --prefix -keys-only

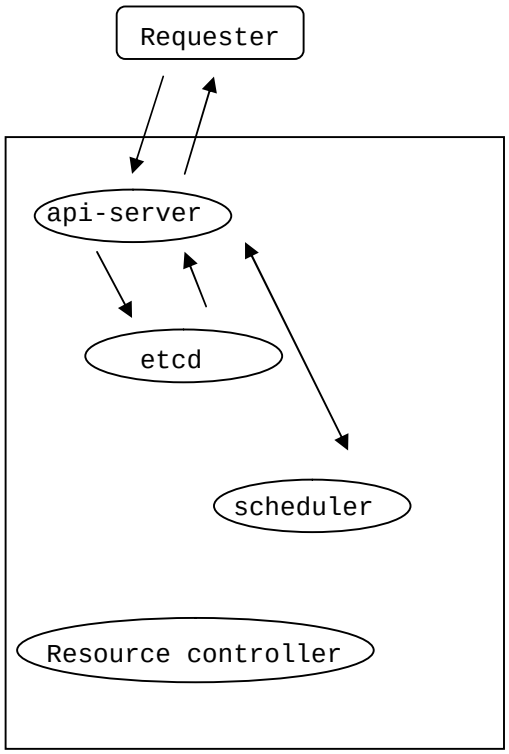
--advertise-client should be configure at kube-api during the setup

ETCD HA Environment

```
etcd.service -->
--initial-cluster controller.0=https://${CONTROLLER0_IP}:2380,
                  controller.1=https://${CONTROLLER0_IP}:2380 \\

################################################################################
```

Kubernetes Architecture



kube-apiserver

Requester --> kube-apiserver --> Authenticate and validate it --> If ok then retrieve data from etcd cluster and response back to requester.

Example adding pod:
curl -x POST /api/v1/namespaces/default/pods...[other]
Backend ->

1. Authentication user done by kube-apiserver

2. Validates the request

3. Retrieve the data

4. Update the ETCD

5. Scheduler continuously monitor kube-apiserver and known new pod is added without node assigned.

6. Scheduler identify right node to add pod and informed it to kube-apiserver.

7. kube-apiserver update the information in etcd cluster.

8. apiserver passed the information to kubelet at appropriate worker node.

9. kubelet create the pod on node and instruct runtime engine to deploy application image, once done then kubelet update the status back to apiserver.

10. apiserver updated back the information in etcd cluster.

Similar pattern run every time change is requested.

kube-apiserver is central point to perform the task.
==========

To view the details how api-server deployed

kubectl get pods -n kube-system

kube-apiserver will shows as pods in output if deployed with kubeadm.

If you deployed with kubeadm - cat /etc/kubernetes/manifests/kube-apiserver.yaml

Non kubeadm setup - cat /etc/systemd/system/kube-apiserver.service
==========

To check apiserver process:

ps -aux | grep -i kube-apiserver
##########

Kube Controller Manager

It is the process which is continuously monitor the status of various components and work towards to make available the system to desired state.

Different types of controllers are Node, Replication, Deployment, Namespace, Endpoint, Job etc..

01. Node Controller:

Command: kubectl get nodes

Node controller --> Monitor the nodes ---> If required taking action through api-server

- Its take the status every 5 seconds.

- If any node is unreachable to node controller then it will wait for 40 seconds before marking it unreachable.

- Once node is mark as unreachable it give 5 minutes to backup.

- Node won't come back within 5 minutes then all pods are remove from unhealthy node and move to healthy one if pods are part of replica sets.

02. Replication Controller:

RC --> API --> Nodes

- It monitor the replica sets and ensure the desired number of pods available every time with in the set, if pods dies it will create another one.

- We can change the node monitor period, grace period and POD evition timeout

```
--node-monitor-period=5s
--node-monitor-grace-period=40s
--pod-eviction-timeout=5m0s
```

- By default all controllers are enable but you can customise as per requirement.

```
--controllers strings
```

Node Monitor Period = 5s
Node Monitor Grace Periods= 40s
Pod eviction Timeout= 5m

==========

To view kube-controller-manager

kubectl get pods -n kube-system

o/p --> If shows controller as container then it installed using kubeadm.
Kubeadm setup: /etc/kubernets/manifests/kube-controller-manager.yaml
Manual: /etc/systemd/system/kube-controller-manager.service

To check kube-controller-manager process:

ps aux | grep kube-controller-manager
##

Kube Scheduler

It responsible pods on nodes. It decide which pod goes on which nodes depends on some criteria-

Scheduler check each pod and decide the best node in two faces -

1. Filter the nodes: It filter the nodes which not fit the profile for the pods.
Example: Node which don't have enough CPU or MEMORY requested by POD so those node are filter out.

2. Rank Nodes: Scheduler calculate total number of resources after placing the pods on node.
Example: Two node, One have 2 CPU and second have 4 CPU are pod placed so in this case our pod will place on second one.

Note: We can customise it and write our own scheduler.

Install kube scheduler:

kubeadm: cat /etc/kubernetes/manifests/kube-scheduler.yaml
Manual: cat /etc/systemd/system/kube

ps -ef | grep kube-scheduler
##

Kubelet

- It is main daemon on worker node.
- It register the node in k8s cluster.

Request--> api server --> kubelet --> Runtime Engine --> Pull Image --> Run the instance (pod)

- Once POD added it monitor it and report to kube-apiserver.
- We can't deploy kubelet using kubeadm have to do it manually on worker node.

To list kubelet process: ps aux | grep kubelet
##

Kube Proxy

- With in k8s cluster every POD can reach to every another POD by using networking solution.
- POD network is internal network with helping of it POD can communicate to each other.

Example: Application and Database POD

a) If application tries to communicate with database POD within the network using ip address it may not get respond as POD IP addresses might change.
b) So in this case we create service which expose DB.
c) The web application can access service:DB.
d) Service also have IP address assigned to it.
e) Service is not container like POD, its virtual components which store in k8s memory.
f) To access service from k8 cluster we use kube proxy.

g) Kube proxy is the process which run on each node in k8s cluster.
h) Once the services created, it will create appropriate rules on each node to forward traffic to services to backend pods.
i) It work like IP tables.

Web --> DB service create --> Proxy Rules on node --> Forward Traffic --> Database

Install kube proxy: Download binaries --> Extract --> Run as service
kubeadm: kubectl get pods -n kube-system / kubectl get daemonset -n kube-system
##

PODs

- Assumptions: Docker image and k8 cluster already setup and its 1:1 setup.

- k8s not deploy containers directly on worker node.

- The containers are encapsulated into k8s object i.e POD.

- POD is single instance of application

- We can scale as per requirement and add new POD in node. 1:1 POD:container if application is same.

- Single POD can have multiple container if application is different.

- Two container can communicate to each other.

Example:

```
docker run python-app
docker run python-app
docker run python-app
docker run python-app

docker run helper -link app01
docker run helper -link app02
docker run helper -link app03
docker run helper -link app04
```

App	Helper	Volume
Python01	App01	Vol01
Python02	App02	Vol02

- With pods k8s deploy everything automatically we just have to define what container consist of and container in the PODs

To deploy PODs

```
kubectl run nginx --image nginx
```

a) kubctl deploy docker container by creating a POD using docker image.
b) nginx image downloaded from docker hub.
c) Can configure k8s to pull image from public or private repository.

To list PODs: kubectl get pods

- In this case external user can't access the nginx web page.

##

Difference Replica Sets/Replication Controller

Because of some reasons POD fails then user can't access their application.
To prevent user to avoid such situation we implement more than one POD on which provide HA.

- Replication controller help in desired state if anyone fails it will deploy another one.
- It also helps in load balancing & scaling.

pod-defination.yaml

```
apiversion: v1
kind: Pod

metadata:
    name: myapp-pod
    labels:
       app: myapp
       type: front-end

spec:
    containers:
    - name: nginx-controller
      image: nginx
```

Remove from metadata to last line and update in replication controller yaml file in the template except apiversion and kind from pod definition file.

Replication controller

1. Its old technologies
2.To create RC

Example: rc-defination.yml

```
apiVersion: v1
kind: ReplicationController
metadata:                          ---> Metadata for Replication Controller
    name: myapp-rc
    labels:
       app: myapp
       type: front-end

spac:                          ---> Spec for Replication Controller

  - template:
       metadata:                   ---> Metadata for POD
           name: myapp-pod
           labels:
              app: myapp
              type: front-end
       spec:                            ---> Spec for POD
           containers:
           - name: nginx-controller
             image: nginx

     replicas: 3
```

Note: template and replicas are child of spec (rc) so it should be in same level.

```
kubectl create -f rc-defination.yaml
```

It first crate POD using pod-defination.yml

```
$ kubectl get replicationcontroller
$ kubectl get pods
################################################################################
```

Replica Sets

replicaset-defination.yml

```
apiVersion: apps/v1     -----> Have to mention app in apiversion else it will fail
kind: ReplicaSet

metadata:                         ---> Metadata for Replica Set
    name: myapp-replicaset    ---> Name of Replica Set
    labels:
       app: myapp
       type: front-end

spac:                         ---> Spec for Replica Set

  - template:
       metadata:                  ---> Metadata for POD
           name: myapp-pod
           labels:
              app: myapp
              type: front-end
       spec:                           ---> Spec for POD
           containers:
           - name: nginx-controller
             image: nginx

     replicas: 3
     selector:
           matchLabels:
                  type: front-end


kubctl create -f replicaset-defination.yml
kubectl get replicaset
```

Note: selector is major difference between Replication controller and Replica set.

a) Selector definition it will help what POD comes under it.
b) We have mention it as replica set also manage the PODs which not comes under it.
Example: There is PODs are created before the creation of replica sets that match labels specified in the selector.
c) The replica set takes those parts also in consideration when creating replicas.

matchLabels selector: Matches the labels which is specified under it to the labels on the pods.

Replica set selector also provide many other options to matches the labels that not available in replication controller.

Labels and Selectors

##

This is default:

```
apiVersion:
kind:
metadata:
```

spec

Thank You