**API Server –** API server is kind of gatekeeper for entire cluster exposes the Kubernetes APIs to nodes, it is the front-end for the Kubernetes control plane.

The Kubernetes API server validates and configures data for the API objects which include pods, services, replication controllers, and others.

We can talk to API sever with kubectl command which is written in go Language

When a user interacts Kubernetes cluster below steps will happens

* Authenticate user
* Validate request
* Retrieve data
* Update ETCD
* Schedule
* kubelet

[root@kubemaster ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-5644d7b6d9-mlmjl 0/1 Running 37 115d

coredns-5644d7b6d9-x8lsz 0/1 Running 37 115d

etcd-kubemaster 1/1 Running 46 115d

**kube-apiserver-kubemaster 1/1 Running 52 115d**

kube-controller-manager-kubemaster 1/1 Running 54 115d

kube-flannel-ds-amd64-rs9tf 1/1 Running 45 114d

kube-flannel-ds-amd64-s5c2n 1/1 Running 11 8d

kube-proxy-ckfqv 1/1 Running 40 115d

kube-proxy-mfnbv 1/1 Running 9 8d

kube-scheduler-kubemaster 1/1 Running 54 115d

[root@kubemaster ~]#

We can see all options related to each service under /etc/kubernetes/manifests

[root@kubemaster manifests]# pwd;ls -lhrt

/etc/kubernetes/manifests

total 16K

-rw------- 1 root root 2.6K Sep 30 20:13 kube-apiserver.yaml

-rw------- 1 root root 2.3K Sep 30 20:13 kube-controller-manager.yaml

-rw------- 1 root root 1.1K Sep 30 20:13 kube-scheduler.yaml

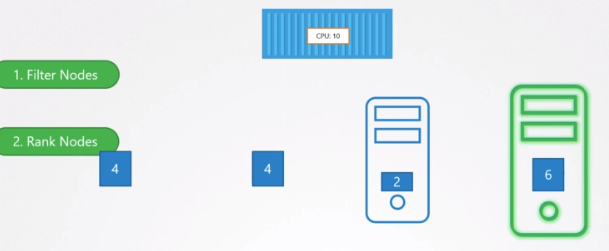
-rw------- 1 root root 1.8K Sep 30 20:13 etcd.yaml

[root@kubemaster manifests]#

**Scheduler –** It is a program on master node which performs the scheduling tasks like launching pods on worker nodes based on resource availability (It maintain resources artifacts in Kubernetes cluster)

Scheduler will always monitors API server for any new pod creations requests. Scheduler will identifies right node for pod deployment and updates back to API servers, API server then update information in ETCD and instructs to kubelet on respective worker node. Kubelet will create pod on worker node and instructs container runtime engine to deploy applications, once done, then kubelet will update back to API server and API server will update data in ETCD.

* Filter nodes
* Rank nodes



We can write our own scheduler section:

* Resource requirements and limits
* Taints and tolerations
* Node selectors/affinity

[root@kubemaster ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-5644d7b6d9-mlmjl 0/1 Running 37 115d

coredns-5644d7b6d9-x8lsz 0/1 Running 37 115d

etcd-kubemaster 1/1 Running 46 115d

kube-apiserver-kubemaster 1/1 Running 52 115d

kube-controller-manager-kubemaster 1/1 Running 54 115d

kube-flannel-ds-amd64-rs9tf 1/1 Running 45 114d

kube-flannel-ds-amd64-s5c2n 1/1 Running 11 8d

kube-proxy-ckfqv 1/1 Running 40 115d

kube-proxy-mfnbv 1/1 Running 9 8d

**kube-scheduler-kubemaster 1/1 Running 54 115d**

[root@kubemaster ~]#

**Controller Manager –** It’s responsible for co-ordination & health of the entire cluster it is going to be conductor and co-ordinator ensuring that nodes are up and running and pods are behaving right way and maintain desired state of the Pods.

**-** There are 4 types of controllers behind this those are:

* Node controller: Responsible for noticing and responding when nodes go down.

Node controller via API servers monitors all nodes every 5 sec (Node Monitor Period = 5s). In case if’s stop receiving heartbeat from a node then it’s mark that as unreachable with in 40s (Node Monitor Grace Period = 40s) . It gives 5 Min to come back node (POD Eviction Timeout = 5m)

In case if node doesn’t come back online with in 5 min then it will removes pods assigned to it and provisions them on healthy nodes if pods are of a replica set one.

* Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
* End point Controller: Populates the Endpoints object that joins Services & Pods
* Service accountant node controller: Create default accounts and API access tokens for new namespaces.

In simple words, controller manager does:

* Watch status
* Remediate situation

there are so many controllers like

* Deployment controller
* Namespace controller
* Job controller PV-Protection controller
* Cronjob
* Stateful-set
* Replica set PV Binder controller
* Node controller
* Replications controller
* Service account controller

…………… and so on

[root@kubemaster ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-5644d7b6d9-mlmjl 0/1 Running 37 115d

coredns-5644d7b6d9-x8lsz 0/1 Running 37 115d

etcd-kubemaster 1/1 Running 46 115d

kube-apiserver-kubemaster 1/1 Running 52 115d

**kube-controller-manager-kubemaster 1/1 Running 54 115d**

kube-flannel-ds-amd64-rs9tf 1/1 Running 45 114d

kube-flannel-ds-amd64-s5c2n 1/1 Running 11 8d

kube-proxy-ckfqv 1/1 Running 40 115d

kube-proxy-mfnbv 1/1 Running 9 8d

kube-scheduler-kubemaster 1/1 Running 54 115d

[root@kubemaster ~]#

**Etcd** – It is a Key value pair data base. It stores the following.

* configuration data of cluster
* Present state of the cluster.
* Each components state of cluster.

Any object can query the etcd to know the cluster state it is single source of the truth of all components & Nodes of the cluster.

Etcd service listens on default 2379 port

# ./etcdctl

**Etcd data store stores information about:**

* Nodes
* Pods
* Configs
* Secrets
* Accounts
* Roles
* Bindings
* others

[root@kubemaster ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-5644d7b6d9-mlmjl 0/1 Running 37 115d

coredns-5644d7b6d9-x8lsz 0/1 Running 37 115d

**etcd-kubemaster 1/1 Running 46 115d**

kube-apiserver-kubemaster 1/1 Running 52 115d

kube-controller-manager-kubemaster 1/1 Running 54 115d

kube-flannel-ds-amd64-rs9tf 1/1 Running 45 114d

kube-flannel-ds-amd64-s5c2n 1/1 Running 11 8d

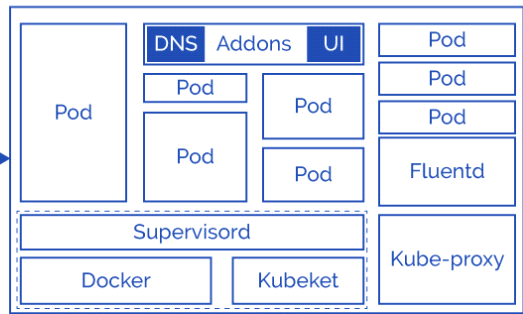
kube-proxy-ckfqv 1/1 Running 40 115d

kube-proxy-mfnbv 1/1 Running 9 8d

kube-scheduler-kubemaster 1/1 Running 54 115d

[root@kubemaster ~]#

**Worker Node components: -**



* Kubelet
* Docker Engine
* Kube-Proxy
* Pod
* fluentd
* Addons

**Kubelet :**

The main Kubernetes agent registers node With Cluster runs on every worker node, talks to API server

Update the configuration of the Node to the etcd of master server.

Initiate pods in worker node based on the specification

Exposes end point on: 10255

* Register node
* Create PODs
* Monitor node & PODs and reports to kube API server on timely basis

[root@**kubeworkernode-1** ~]# ps -aux | grep -i kubelet

root 684 9.7 7.6 931160 79124 ? Ssl 19:33 11:57 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --cgroup-driver=systemd --network-plugin=cni --pod-infra-container-image=k8s.gcr.io/pause:3.1

root 31671 0.0 0.0 112716 984 pts/0 S+ 21:35 0:00 grep --color=auto -i kubelet

[root@kubeworkernode-1 ~]#

**Docker Engine –** It’s a platform to run containers present on each Worker Node, pulling images, stopping and starting containers.

Kublet & Docker Engine together part of supervisord layer/process.

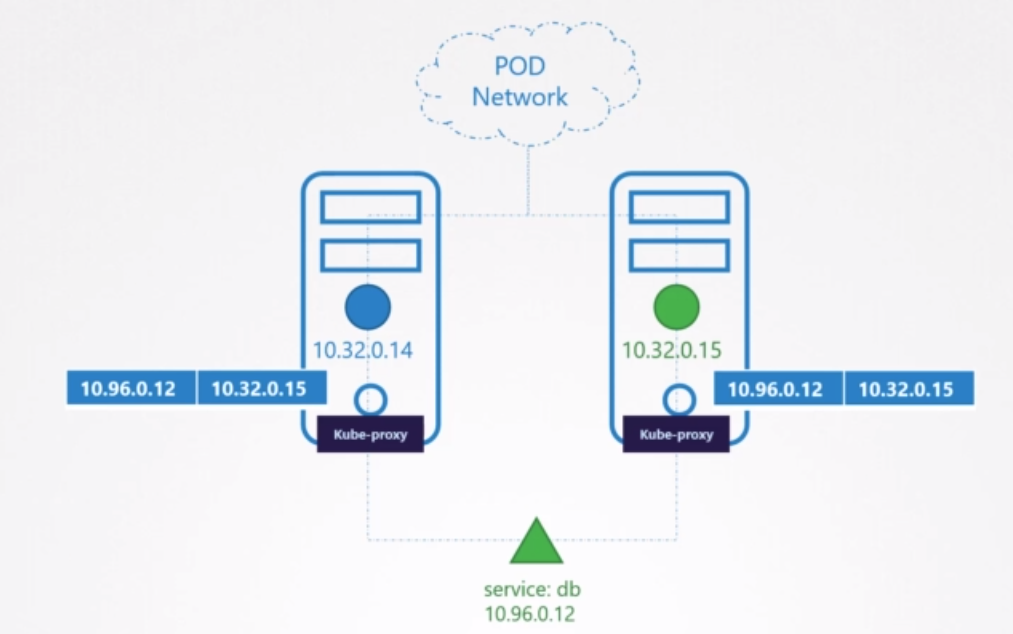
**Kube-Proxy –**It is core networking element responsible for entire network configuration & management. It routes the traffic to appropriate containers based on IP address and port number of the incoming request. In other words, we can say it is used for port translation.

[root@**kubeworkernode-1** ~]# ps -aux | grep -i kube-proxy

root 2498 0.2 2.3 142600 23868 ? Ssl 19:36 0:21 /usr/local/bin/kube-proxy --config=/var/lib/kube-proxy/config.conf --hostname-override=kubeworkernode-1

root 31947 0.0 0.0 112716 984 pts/0 S+ 21:37 0:00 grep --color=auto -i kube-proxy

[root@kubeworkernode-1 ~]#



[root@kubemaster manifests]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

coredns-5644d7b6d9-mlmjl 0/1 Running 37 115d

coredns-5644d7b6d9-x8lsz 0/1 Running 37 115d

etcd-kubemaster 1/1 Running 46 115d

kube-apiserver-kubemaster 1/1 Running 52 115d

kube-controller-manager-kubemaster 1/1 Running 54 115d

kube-flannel-ds-amd64-rs9tf 1/1 Running 45 114d

kube-flannel-ds-amd64-s5c2n 1/1 Running 11 8d

**kube-proxy-ckfqv 1/1 Running 40 115d**

**kube-proxy-mfnbv 1/1 Running 9 8d**

kube-scheduler-kubemaster 1/1 Running 54 115d

[root@kubemaster manifests]#

[root@kubemaster manifests]# kubectl get daemonset -n kube-system

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE

kube-flannel-ds-amd64 2 2 2 2 2 <none> 115d

kube-flannel-ds-arm 0 0 0 0 0 <none> 115d

kube-flannel-ds-arm64 0 0 0 0 0 <none> 115d

kube-flannel-ds-ppc64le 0 0 0 0 0 <none> 115d

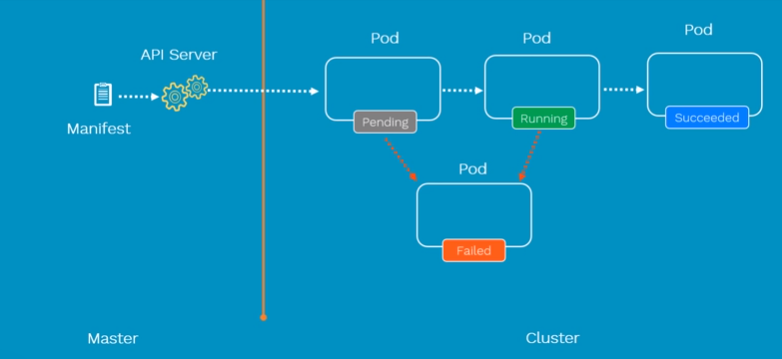
kube-flannel-ds-s390x 0 0 0 0 0 <none> 115d

**kube-proxy 2 2 2 2 2 beta.kubernetes.io/os=linux 115d**

[root@kubemaster manifests]#

**PODS & POD deployment: -**

Pod is nothing but basic deployment unit contains one more container You can see the attached pod life-cycle & pod-network & pod-manifest file



**Create the Pod With yaml :**

[root@kubemaster yamls-self]# cat nginx.yaml

**apiVersion:** v1 or apps/v1

**kind:** Pod service

**metadata:**

name: nginx-pod

labels:

app: nginx

env: test

**spec:**

containers:

- name: nginx-container

image: nginx

[root@kubemaster yamls-self]#

* kubectl create -f nginx.yaml //To create the pods
* kubectl get pods //To get pods information
* kubectl get pods -o wide //To get pods information wide
* kubectl get pod nginx-pod -o yaml //To get the pod information in the form of yaml
* kubectl describe pod nginx-pod //To describe the target pod
* kubectl explain pods //Give the information to build the pods like apliversions, kind etc
* **Ex:** to check the pod status
* kubectl exec -it nginx-pod sh //To connect to the target pod

**Create the Pod on Specific Node: -**

You can create the pod with specific node with the help of "nodeSelector" object in Kubernetes

kubectl describe node <node name> //Give all configuration of the target node.

**Ex**: kubectl describe node kubeworker1

**Manifesto file for create the pod in targeted node:**

[root@kubemaster Kubernates]# cat nodeselectoringinx.yml

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

env: test

spec:

nodeSelector:

kubernetes.io/hostname: kubeworker2 //Node label should be given `

containers:

**-** name: nginx-container

image: nginx

ports:

**-** containerPort: 80

**Explore with below commands: -**

* kubectl create -f nodeselectoringinx.yml
* kubectl get pods -o wide
* kubectl delete -f nodeselectoringinx.yml

**Replication Controller or rc :-**

Replication controller ensures that a specified number of pods running any time it provides high availability. If any pods are died replication controller creates new pods with same specification on same node or other node if node itself dies.

* In vice versa if any excess pods found it will be killed. Replication controller and pods are associated with "labels" it supports the equality based selectors ( =, ==,!= )
* Replication controller object is old it is replaced with Replica set.
* The difference is Replication controller support equality-based selectors replica set supports equality & set based selectors ( in, notin, exists )
* Replication Controller: Ensure that High availability of the Pods
* Replica Sets are Next generation of replica controllers

**Replication controller manifest file & creation**

[root@kubemaster yamls-self]# cat tomcat.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: tomcat-rc

spec:

replicas: 3

template:

metadata:

name: tomcat-rc

labels:

app: tomcat

env: test

spec:

containers:

- name: tomcat-rc

image: tomcat

ports:

- containerPort: 7474

[root@kubemaster yamls-self]#

[root@kubemaster kube-Deploy] # kubectl create -f nginxrc.yaml //To create the rc

[root@kubemaster kube-Deploy]# kubectl get rc //To Get the status of rc

[root@kubemaster kube-Deploy]# kubectl get rc -o wide //To Get the status of rc in wide

[root@kubemaster kube-Deploy]#kubectl get pods //To Get the status of Pods

[root@kubemaster kube-Deploy]# kubectl get pods -o wide //To Get the status of Pods wide

[root@kubemaster kube-Deploy]# kubectl get pods -l app=nginx-app //To Get the status of Pods with label

[root@kubemaster kube-Deploy]# kubectl describe rc nginx-rc //To Get the complete information of the rc

**Test the rc :**

kill the pod see the status of pods

shutdown the node and see the status of pods

Scale up & auto scale the app instance:

kubectl scale rc <rcname> --replicas=<number of replicas> //To do the scale of replicas

Ex: [root@kubemaster]# kubectl scale rc nginx-rc --replicas=5 //To increase/scaleup the number of pods