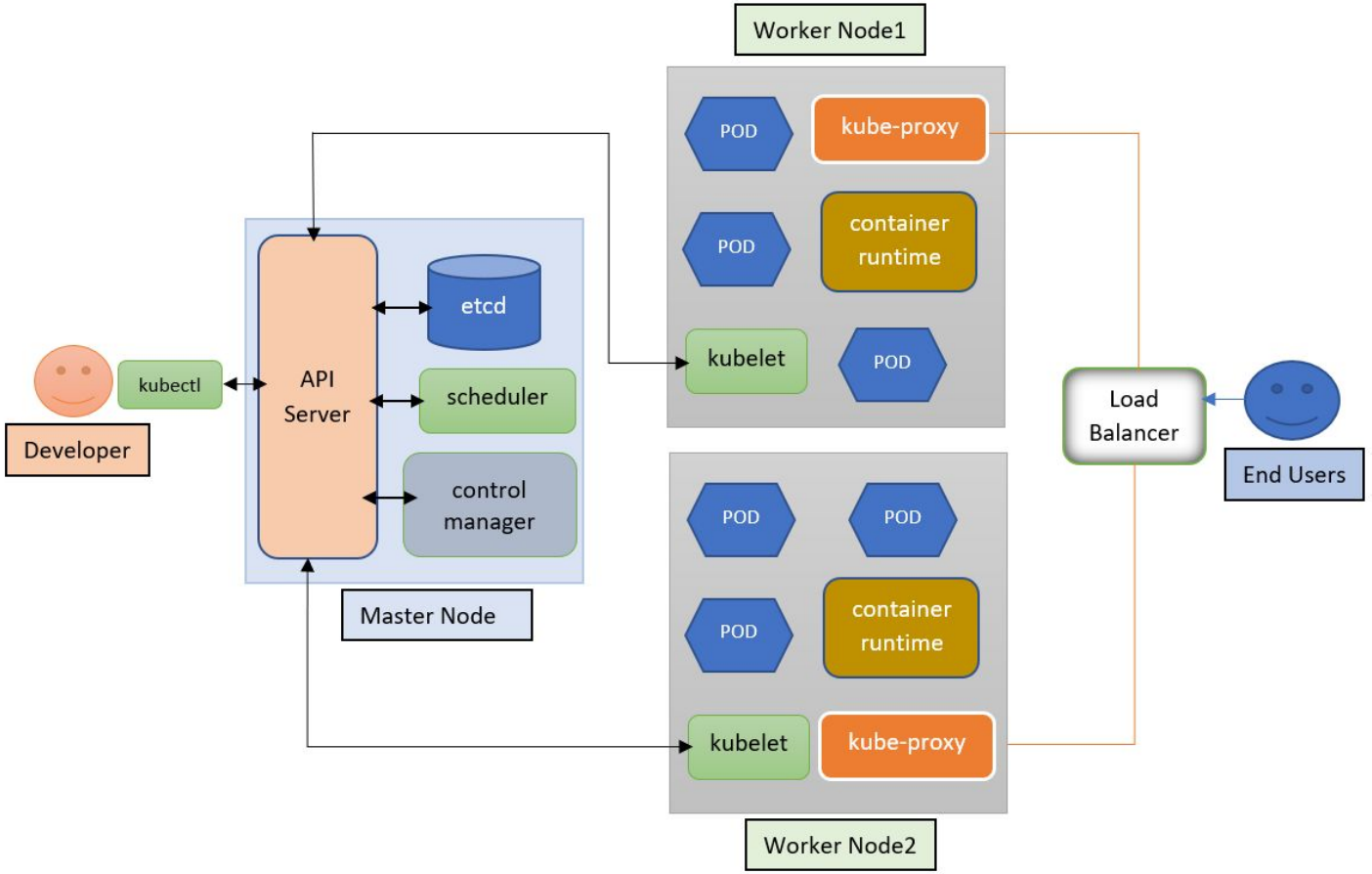
Kubernetes

By Praveen Singampalli

Instagram/Telegram/twitter – SINGAM4DEVOPS

Kubernetes Architecture:

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
The worker node(s) host the Pods that are the components of the application workload.
The control pane manages the worker nodes and the Pods in the cluster.



Components of Kubernetes:

- **Master Node Components:**

1. API Server 2. Controller Manager 3. ETCD 4. Scheduler

1) API Server:

It is the front-end for the Kubernetes control plane.

2) Controller Manager: This is a component on the master that runs controllers.

- **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, it joins Services and Pods).
- **Service Account and Token Controllers:** Create default accounts and API access tokens for new namespaces.

3) **ETCD**: Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

4) **kube-scheduler -** Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment

1) **Kubelet** - An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

2)**kube-proxy** - kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. Kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

3)**Container runtime** - The container runtime is the software that is responsible for running containers.

Manifest File Components:

apiversion

Kind

Metadata

Spec

File name : nginx-deployment.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Service components:

Kubernetes ServiceTypes allow you to specify what kind of Service you want. The default is ClusterIP.

Type values and their behaviors are:

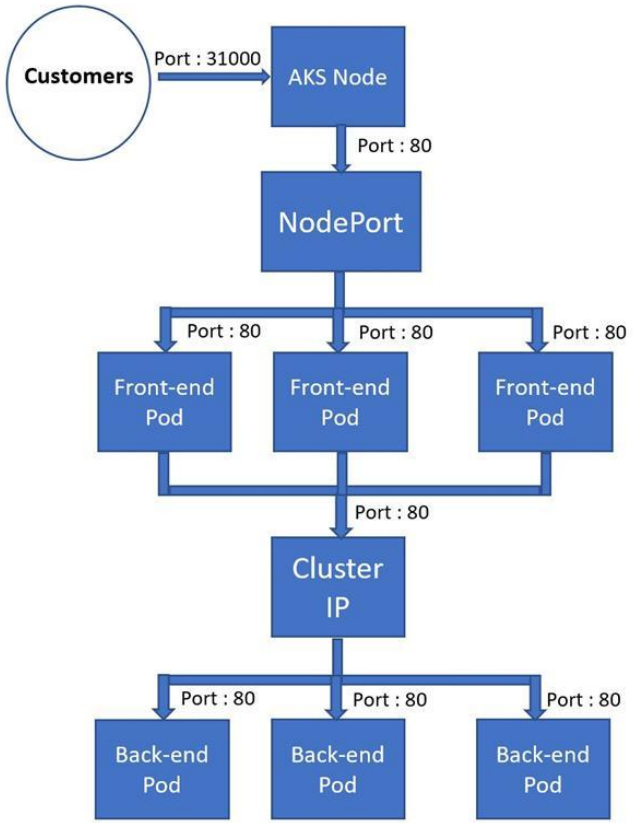
ClusterIP: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.(To talk to other nodes in the cluster)

NodePort: Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>. (The entrpoint for node)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
      # By default and for convenience,
    - port: 80
      targetPort: 80
      # Optional field
      # By default and for convenience,
      nodePort: 30007
```

- **LoadBalancer**: Exposes the Service externally using a cloud provider's load
balancer. NodePort and ClusterIP S ervices, to which the external load balancer routes, are automatically created.
- **Externalname**: Maps the Service to the contents of
the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```




```yaml
kind: Service
apiVersion: v1

metadata:
  name: hostname-service

spec:
  type: NodePort
  selector:
    app: echo-hostname
  ports:
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

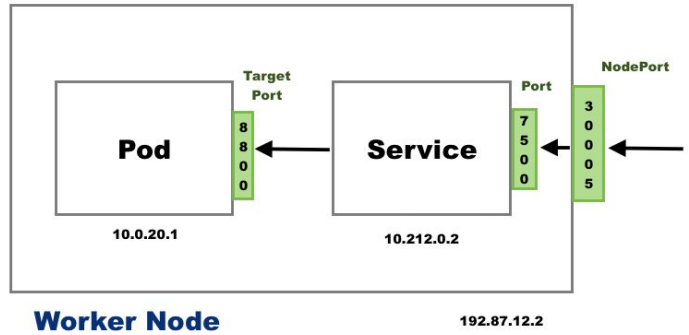
Make the service available to network requests from external clients

Forward requests to pods with label of this value

`nodePort`
access service via this external port number

`port`
port number exposed internally in cluster

`targetPort`
port that containers are listening on



Routing In kubernetes

Kubernetes Commands

1) Kubectl create deploy my-nginx –image=nginx –dry-run=client –o yaml > mynginx.yaml
2) Kubectl get pods
3) Kubectl get namespace
4) Kubectl create ns mydev
5) Kubectl create –f pod.yaml
6) Kubectl describe svc nginx
7) Kubectl exec -it pod1 bash

Kubernetes IQ:

1) What is the architecture of kubernetes
2) What does control manager, etcd, scheduler, API server do
3) What is a manifest file and what are the components of it
4) What is node affinity, pod afiinity , taint toleration
5) What is node port, cluster ip
6) What is persitant volumes and why we use it
7) Describe what is pod and what is pod lifecycle
8) What are the components on master and worked node
9) What is ingress controller
10) What are types of services in kuberntes
11) How one pod talks with other pod
12) How the pod healthcheck is done(describe rediness, livesness)
13) How the monitoring is done(integration on Prometheus and grafana)
14) What is deamonset, replicaset, horizontal pod autoscaler
15) Write a manifest file of your own choice
16) What is namespace and why we use it
17) What are helm charts and uses