Kubernetes

An Orchestration Tool

Agenda

- **Introduction**
  - Who am I?
  - What is Kubernetes?
  - What does kubernetes do?
- **Architecture**
  - Master Components
  - Node Components
  - Additional Servers
  - Networking
- **Concepts**
  - Core
  - Workloads
  - Network
  - Storage
  - Configuration
  - Auth and Identity
- **Behind the Scenes**
  - Deployment from Beginning to End

Introduction

Who Am I ?

Sandeep Anuragi / blackmagiclinux@gmail.com

Github: @darwikdev1

Cloud Administrator @ Training Basket

Red Hat Instructor

What is Kubernetes ?

Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.

What Does Kubernetes do ?

- Kubernetes is the linux kernel of distributed system.

  It abstracts away the underlying hardware of the nodes and provides a uniform interface for applications to the both deployed and consume the shared pool of resources.

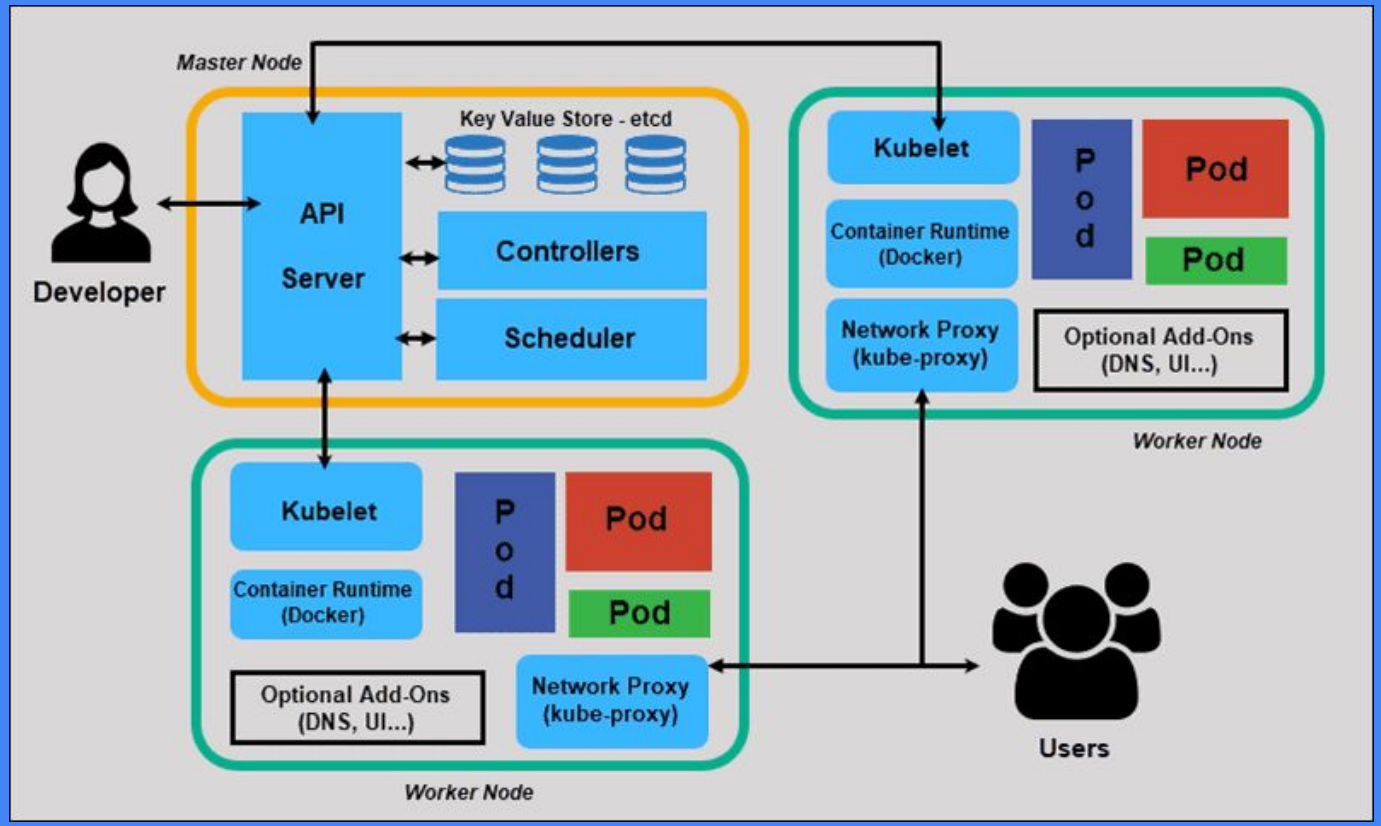
Kubernetes Architecture

Architecture Overview

Masters: Acts as the primary Control Plane for kubernetes. Masters are responsible at a minimum for running the API server, scheduler, and cluster controller. They commonly also manage storing cluster state, cloud-provider specific components and other cluster essential services.

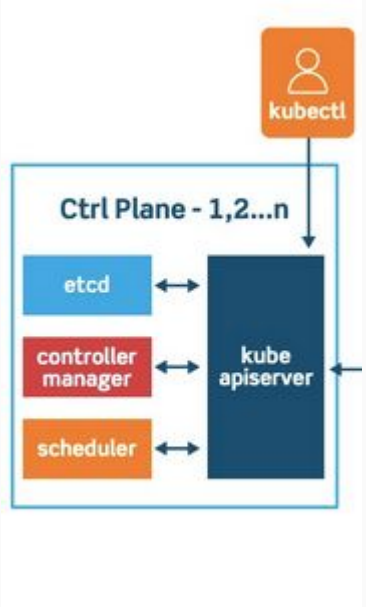
Nodes: Are the ‘workers’ of a kubernetes cluster. They run a minimal agent that manages the node itself, and are tasked with executing workloads as designated by the master.

Architecture Diagram



Master Components

Master Components



- Kube-apiserver
- Etcd
- Kube-controller-manager
- cloud-controller-manager
- kube-scheduler

kube-apiserver

The apiserver provides a forward facing REST interface into the kubernetes control plane and datastore. All clients, including nodes, users and other applications interact with kubernetes strictly through the API Server.

It is the true core of kubernetes acting as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

Etcd

Etcd acts as the cluster datastore; providing a strong, consistent and highly available key-value store used for persisting cluster state.

Kube-controller-manager

The controller-manager is the primary daemon that manages all core component control loops. It monitors the cluster state via the apiserver and steers the cluster towards the desired state.

Cloud-controller-manager

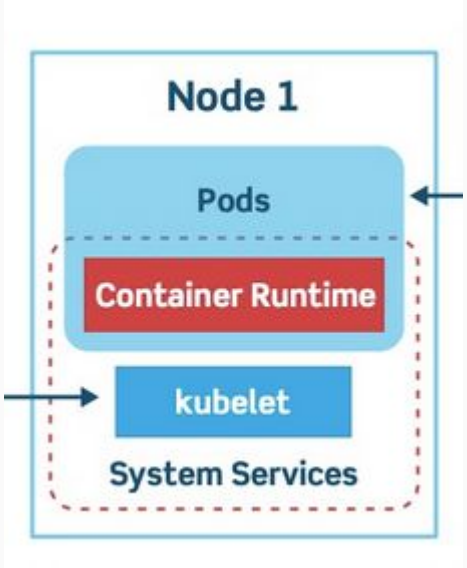
The **cloud-controller-manager** is a daemon that provides cloud-provider specific knowledge and integration into the core control loop of kubernetes. The controllers include Node, Route, Service, and add an additional controller to handle **PersistentVolumeLabels.**

Kube-scheduler

Kube-scheduler is a verbose policy-rich engine that evaluates workload requirements and attempts to place it on a matching resources. These requirements can include such things as general hardware reqs, affinity, anti-affinity, and other custom resource requirements.

Node Components

Node Components



- kubelet
- Kube-proxy
- Container runtime engine

Kubelet

Acts as the node agent responsible for managing pod lifecycle on its host.

Kubelet understands YAML container manifests that it can read from several sources.

- File path
- HTTP endpoint
- Etcd watch acting on any changes
- HTTP server mode accepting container manifests over a simple API.

kube-proxy

Manages the network rules on each node and performs connection forwarding or load balancing for kubernetes cluster services.

Available Proxy Modes:

- Userspace
- IPtables
- ipvs (alpha in 1.8)

Container Runtime

With respect to kubernetes, A container runtime is a CRI (container runtime interface) compatible application that executes and manages containers.

- Containerd (docker)
- Cri-O
- Rkt
- Kata (formerly clear and hyper)
- Virtlet (VM CRI compatible runtime)

Additional Services

Kube-dns : Provides cluster wide DNS services. Services are resolvable to

<service>.<namespace>.svc.cluster.local.

Heapster: Metrics collectors for kubernetes cluster, used by some resources such as the horizontal pod autoscaler. (required for kube-dashboard metrics)

Kube-dashboard: A general purpose web based UI for kubernetes.

Networking

Networking - Fundamental Rules

- All pods can communicate with all others Pods without NAT
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as is the same IP the other see it as.

Networking - Fundamentals Applied

Containers: in a pod exist within the same network namespace and share an IP; allowing for intrapod communication over localhost.

Pods are given a cluster unique IP for the duration of its lifecycle, but the pods themselves are fundamentally ephemeral.

Services are given a persistent cluster unique IP that spans the pod lifecycle.

External Connectivity is generally handled by an integrated cloud provider or other external entity (load balancer)

Networking - CNI

Networking within kubernetes is plumbed via the container network interface (CNI), an interface runtime and a network implementation plugin.

Compatible CNI network plugins.

calico	**Kube-router**
cillium	**multus**
contiv	**openVswitch**
GCE	**ovn**
Flannel	**weave**

Kubernetes Concepts

Kubernetes Concepts - Core

Cluster - A collection of hosts that aggregate their available resources including cpu, ram, disk, and their devices into a usable pool.

Master - the master(s) represent a collection of components that make up the control plane of kubernetes. These components are responsible for all cluster decisions including both scheduling and responding to cluster events.

Node - A single host, physical or virtual capable of running pods. A node is managed by the master(s), and at a minimum runs both kubelet and kube-proxy to be considered part of the cluster.

Namespace - A logical cluster or environment. Primary method of dividing a cluster or scoping access.

Concepts - core

Label - key-value pairs that are used to identify, describe and group together related sets of objects. Labels have a strict syntax and available character set.

Annotation - key-value pairs that contain non-identifying information or metadata. Annotations do not have the syntax limitations as labels and can contain structured or unstructured data.

Selector - selectors use label to filter or select objects. Both equality-based(=,==,!=) or simple key-value matching selectors are supported.

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name:  nginx
  annotations:
    description: "nginx frontend"
  labels:
    app: nginx
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      tier: frontend
  template:
    metadata:
      labels:
        app: nginx
        tier: frontend
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       ports:
       - containerPort: 80
```

Labels, and Annotations, and Selectors

Labels:
app: nginx
tier: frontend

Annotations:
description: “nginx frontend”

Selector:
app: nginx
tier: frontend

Set based selectors

Valid Operators:

- In
- Notin
- Exists
- DoesNotExist

Supported Objects with set-based selectors:

- Job
- Deployment
- ReplicaSet
- DaemonSet
- PersistentVolumeClaims

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx
  annotations:
    description: "nginx frontend"
  labels:
    app: nginx
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      - {key: app, operator: In, values: [nginx]}
      - {key: app, operator: In, values: [nginx]}

  template:
    metadata:
      labels:
        app: nginx
        tier: frontend
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       ports:
       - containerPort: 80
```

Concepts - Workloads

Pod - A pod is the smallest unit of work or management resource within kubernetes. It is comprised of one or more containers that share their storage, network, and context (namespace, cgroup etc).

ReplicationController - Method of managing pod replicas and their lifecycle. Their scheduling, scaling and deletion.

ReplicaSet - Next Generation ReplicationController. Supports set-based selectors.

Deployment - A declarative method of managing stateless Pods and ReplicaSets. Provides rollback functionality in addition to more granular update control mechanism.

Deployment

Contains configuration of how updates or ‘deployments’ should be managed in addition to the pod template used to generate the ReplicaSet.

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name:  nginx
  annotations:
    description: "nginx frontend"
  labels:
    app: nginx
    tier: frontend
spec:
  replicas: 3
  mindReadySeconds: 10
  Strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 5
      maxUnavailable: 2
  Selector:
    matchLabels:
      app: nginx
      tier: frontend
  template:
    metadata:
      labels:
        app: nginx
        tier: frontend
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       ports:
       - containerPort: 80
```

ReplicaSet

Generated ReplicaSet from Deployment Spec.

```yaml
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: nginx
  annotations:
    description: "nginx frontend"
  labels:
    app: nginx
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      tier: frontend
  template:
    metadata:
      labels:
        app: nginx
        tier: frontend
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       ports:
       - containerPort: 80
```

Concepts - Workload (cont.)

StatefulSet - A controller tailored to managing pods that must persist or maintain state. Pod identity including hostname, networking, and storage will be persisted.

DeamonSet - Ensures that all nodes matching certain criteria will run an instance of a supplied pod. Ideal of cluster wide services such as log forwarding, or health monitoring.

StatefulSet

- Attaches to ‘headless service’ (not shown) nginx.
- Pod given unique ordinal names using the pattern

  <statefulset name>-<ordinal index>.

- Creates independent persistent volumes based on the

  ‘volumeClaim Templates’.

```
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: nginx
spec:
  servicename: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       ports:
       - containerPort: 80
         name: web
       volumeMounts:
       - name: www
         mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
      requests:
      storage: 1Gi
```

DaemonSet

- Bypass default scheduler
- Schedulers a single instance on every host while adhering to tolerances and taints.

```yaml
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: nginx
  namespace: kube-system
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
          name: web
```

Concepts - workloads (cont.)

Job - the job controller ensures one or more pods are executed and successfully terminates. It will do this until it satisfies the completion and/or parallelism condition.

CronJob - An extension of the jobController, it provides a method of executing jobs on a cron-like schedule.

Jobs

- Number of pod executions can be controlled via spec.completions
- Jobs can be parallelized using spec.parallelism
- Jobs and Pods are NOT automatically cleaned up after a job has completed.

```
apiVersion: apps/v1beta2
kind: Job
metadata:
  name: hello
spec:
  completions: 10
  parallelism: 2
  template:
    metadata:
      name: hello
    spec:
      containers:
      - name: hello
        image: alpine:latest
        command: ["echo", "hello there!"]
      restartPolicy: Never
  backoffLimit: 4
```

CronJob

- Adds cron schedule to job template

```
apiVersion: apps/v1beta2
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "30 8 * * * "
  jobTemplate:
    spec:
      template:
        metadata:
          name: hello
   spec:
     containers:
     - name: hello
       image: alpine:latest
       command: ["echo", "hello there!"]
     restartPolicy: Never
```

Concepts - Network

Concepts - Network

Service : service provide a method of exposing and consuming L4 pod network accessible resource. They use label selectors to map groups of pods and ports to a cluster-unique virtual IP.

Ingress : An ingress controller is the primary method of exposing a cluster service (usually http) to the outside world. These are load balancers or routers that usually offer SSL termination, named-based virtual hosting etc.

Service

- Acts a unified method of accessing replicated pods.
- Four major Service Types:
  - **ClusterIP** exposes service on a strictly cluster-internal IP (default)
  - **NodePort** service is exposed on each node's IP on a statically defined port.
  - **LoadBalancer** works in combination with a cloud provider to expose a service outside the cluster on a static external IP
  - **ExternalName** used to references endpoints OUTSIDE the cluster by providing a static internally referenced DNS name.

```
kind: Service
apiVersion: v1
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Ingress

- Deployed as a pod to one or more hosts
- Ingress controllers are an external controller with multiple options.
  - Nginx
  - HAproxy
  - Contour
  - Traefix
- Specific features and controller specific configuration is passed through annotations.

```
kind: Ingress
apiVersion: v1
metadata:
  annotations:
  kubernetes.io/ingress.class: "nginx"
  name: nginx
spec:
  rules:
  - hosts: foo.bar.com
    http:
      paths:
      - path: /nginx
        backend:
          service: nginx
          servicePort: 80
```

Concepts - Storage

Volume - storage that is tied the pod lifecycle, consumable by one or more containers within the pod.

PersistentVolume - A PersistentVolume (PV) represents a storage resource. PVs are commonly linked to a backing storage resource. NFS, GCEPersistentDisk, RBD etc. and are provisioned ahead of time. Their lifecycle is handled independently from a pod.

PersistentVolumeClaim - A PersistentVolumeClaim(PVC) is a request for storage that satisfies a set of requirements instead of mapping to a storage resource directly. Commonly used with dynamically provisioned storage.

StorageClass - Storage classes are an abstraction on top of an external storage resource. These will include a provisioner configuration parameters as well as a PV reclaimPolicy.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: www
  volumes:
  - name: www
    emptyDir: {}
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: www
  volumes:
  - name: www
    awsElasticBlockStore:
      volumeID: <volume-id>
      fsType: ext4
```

Persistent Volumes

- PVs are a cluster-wide resource
- Not directly consumable by a Pod
- PV Parameters:
  - Capacity
  - accessModes
    - ReadOnlyMany (ROX)
    - ReadWriteOnce (RWO)
    - ReadWriteMany(RWX)
  - persistentVolumeReclaimPolicy
  - StorageClass

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs
spec:
  capacity:
    storage: 500Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  PersistentVolumeReclaimPolicy: Recycle
  StorageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /data
    server: 10.255.100.10
```

Persistent Volume Claims

- PVCs are scoped to namespaces
- Supports accessModes like PVs
- Uses resource request Model similar to Pods
- Claims will consume Storage from matching PVs or StorageClasses based on storageClass and selectors.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-nfs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Gi
  storageClass: slow
```

Storage Classes

- Uses an external system defined by the provisioner to dynamically consume and allocate storage.
- Storage class fields
  - Provisioner
  - Parameters
  - reclaimPolicy

```yaml
apiVersion: v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
reclaimPolicy: Delete
parameters:
  monitors: 10.16.153.105:6789
  adminId: kube
  adminSecretName: ceph-secret
  adminSecretNamespace: kube-system
  pool: kube
  userId: kube
  userSecretName: ceph-secret-user
  fsType: ext4
  imageFormat: "2"
```

Concepts - configuration

ConfigMap - Externalized data stored within kubernetes that can be referenced as a command-line argument, environment variable, or injected as a file into a volume mount. Ideal for separating containerized application from configuration.

Secret - Functionally identical to ConfigMaps, but stored encoded as base64, and encrypted at rest (if configured).

- Can be used in pod config:
  - Injected as a file
  - Passed as an environment variable
  - Used as a container command (requires passing as env var)

```yaml
apiversion: v1
kind: ConfigMap
metadata:
  name: my-cm
data:
  name: mydata.txt
  contents:
    you can store
    multiline content
    and configfiles
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - image: nginx:latest
      name: nginx
      volumeMounts:
        - name: myConfigMaps
          path: /etc/config
  volumes:
  volumes:
    - name: myConfigMap
      configMap:
        name: my-cm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - image: nginx:latest
      name: nginx
      env:
        - name: USERNAME
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: username
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: nginx
data:
  username: aGvycGGRlcnA=
  password: aq2ei3rh@HUH#VV
```

Concepts - Auth and Identity (RBAC)

[Cluster]Role - Roles contain rules that act as a set of permissions that apply verbs like ‘get’, ‘list’, ‘watch’ etc over resources that are scoped to apiGroups. Roles are scoped to namespaces, and ClusterRoles are applied cluster-wide.

[Cluster]RoleBinding - Grant the permissions as defined in a [cluster]role to one or more “subjects” which can be a user, group, or service account.

ServiceAccount - ServiceAccounts provides a consumable identity for pods or external services that interact with the cluster directly and are copied to namespaces.

[Cluster]Role

- Permissions translate to url path. With “” defaulting to core group.
- Resources act as items the role should be granted access to.
- Verbs are the actions the role can perform on the referenced resources.

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: monitor-things
rules:
- apiGroups: [""]
  Resources: ["services","endpoints","pods"]
  verbs: ["get","list","watch"]
```

[Cluster]RoleBinding

- Can reference multiple subjects
- Subjects can be of kind:
  - User
  - Group
  - ServiceAccount
- roleRef targets a single role only.

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: monitor-things
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: monitor-things
  apiGroup: rbac.authorizaton.k8s.io
```

Behind the scenes