Kubernetes (k8s)

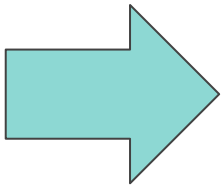
Rajesh G

CTO, Managing Partner
https://unigps.in

Training Objectives

At the end of training,

participants should be able to



- ❏ Know Kubernetes and Be a Helmsman
- ❏ Create and run PODs
- ❏ Bundle applications & Deploy
- ❏ Service apps using Load Balancers
- ❏ Troubleshoot

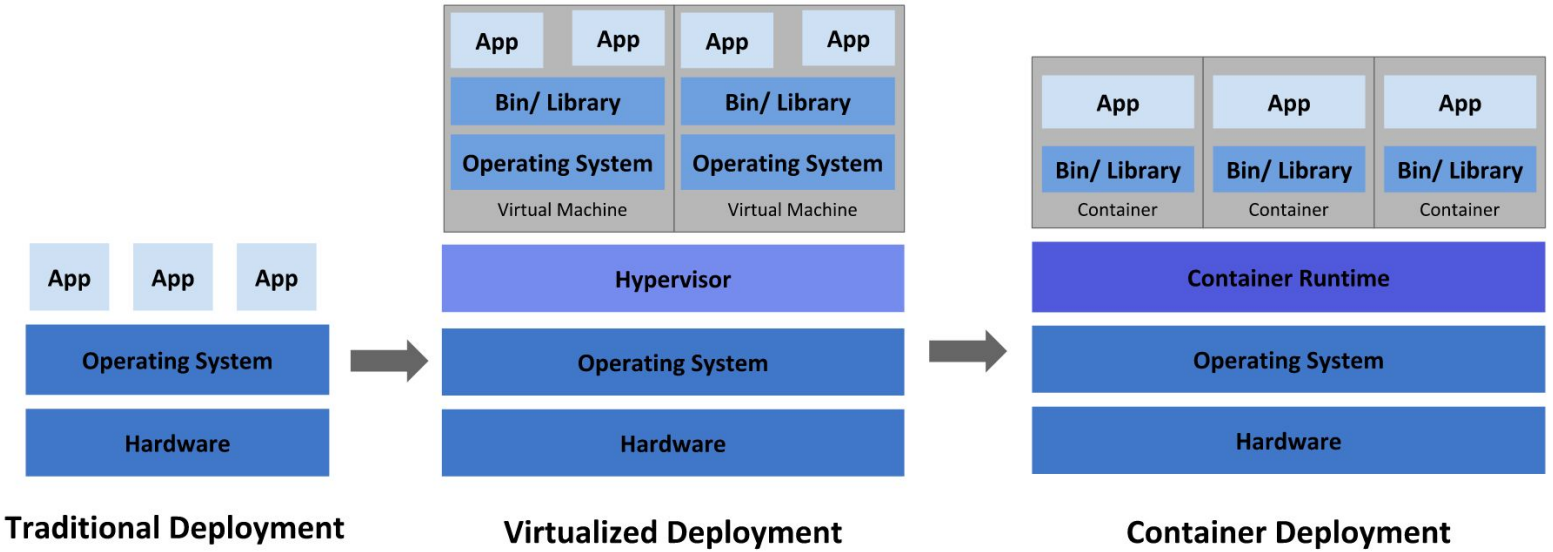
Table of Contents

Table of Contents

Module 1: Overview

- What is Kubernetes
- Architecture
- Components
  - Master Components
  - Node Components
  - Add ons
- Objects

Deployment - Journey



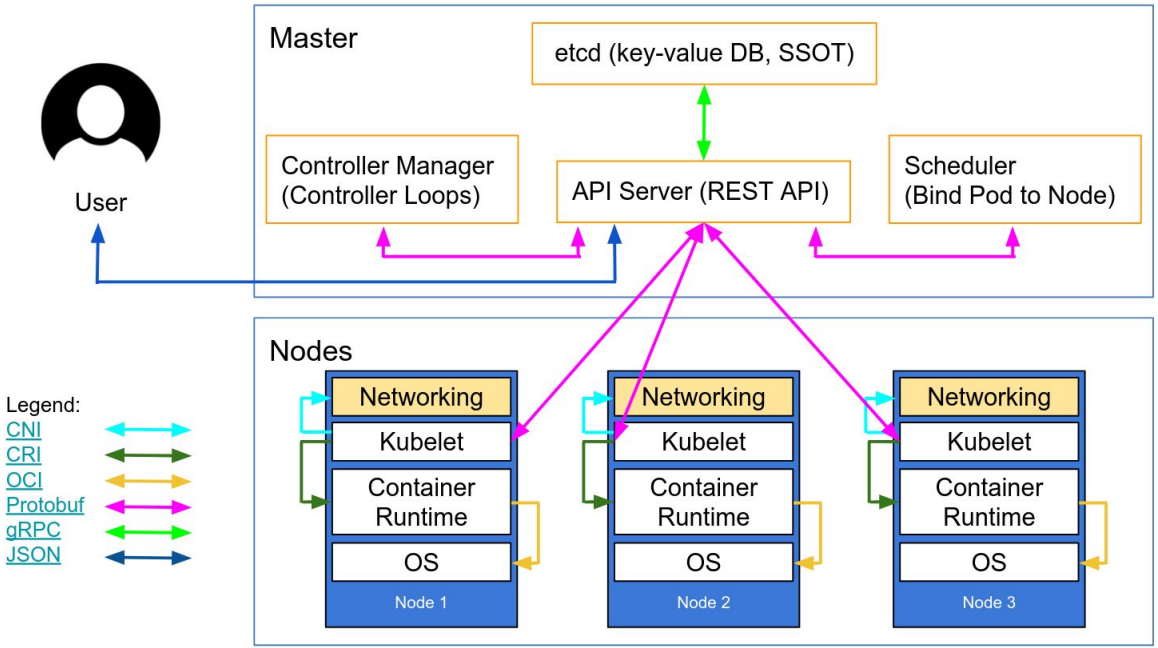
What is / why Kubernetes

Kubernetes - Helmsman (in ancient greek): Guy who steers ship / boat

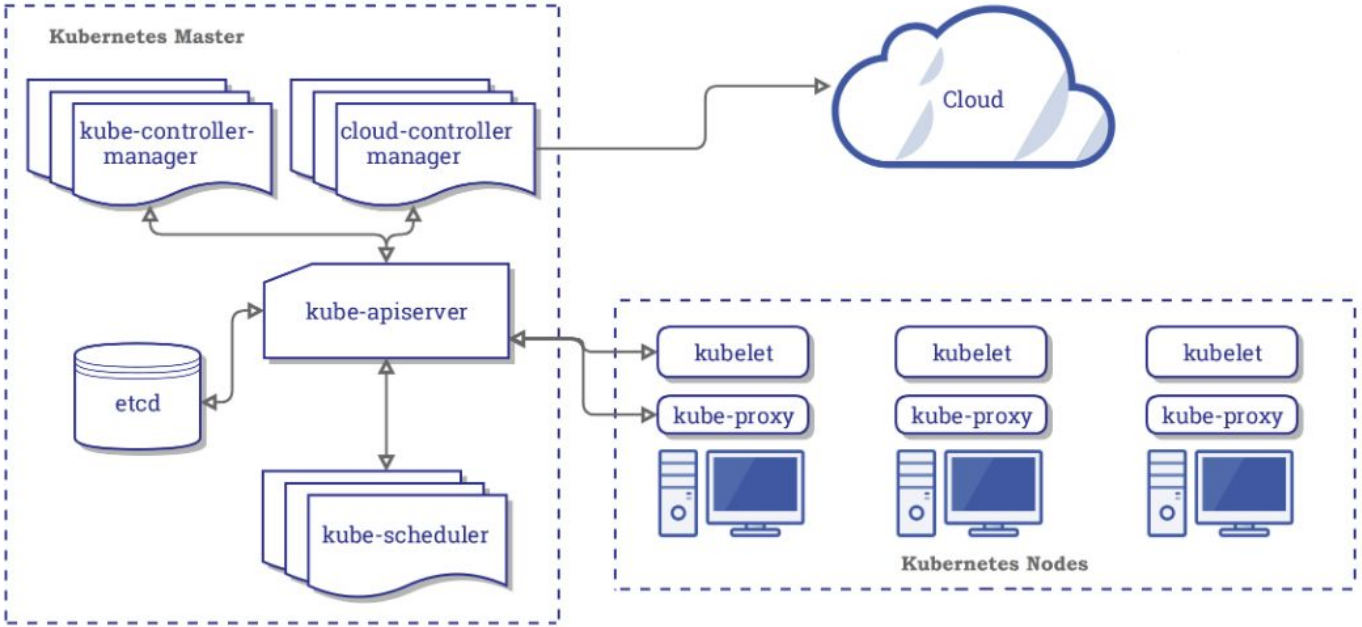
Why Kubernetes?

- Service Discovery & Load Balancing
- Storage Orchestration
- Automated rollouts & rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration Management

Architecture - Overview



Architecture (view 2)



Master Components - ETCD

- Distributed reliable key-value store that is simple, secure & fast
- Uses RAFT based consensus algorithm to work in distributed environment
- Key value store distributed database
- Runs on port 2379

Master Components - API Server

- The central management entity
- Only component that connects to ETCD
- Designed for horizontal scaling

Connectivity:

- External: kubectl
- Internal: kubelet
- Persistent Storage: ETCD

Master Components - Scheduler

Schedules pods on appropriate Node(s)

Watches for newly created PODs that have no nodes assigned

Decision Parameters:

- Resource requirements (memory, cpu, disk type say SSD)
- Hardware, Software, Policy requirements
- Affinity, Anti-affinity
- Data locality
- Inter workload interference
- Deadlines

Master Components - Kube Controller

- 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, joins Services & Pods)
- Service Account & token Controller
  - Create default accounts and API access tokens for new namespaces

Master Components - Cloud Controller

- Route Controller
  - For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Service Controller
  - For setting up routes in the underlying cloud infrastructure
- Service Controller
  - For creating, updating and deleting cloud provider load balancers
- Volume Controller
  - For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

Node Components - kubelet

- Runs on every node
- Ensures containers are running & healthy in PODs
- Doesn’t manage container not created by K8S

Node Components - kube-proxy

- Network proxy that runs on every node in cluster
- Maintains network rules on nodes
- Uses OS packet filtering layer else forwards traffic itself

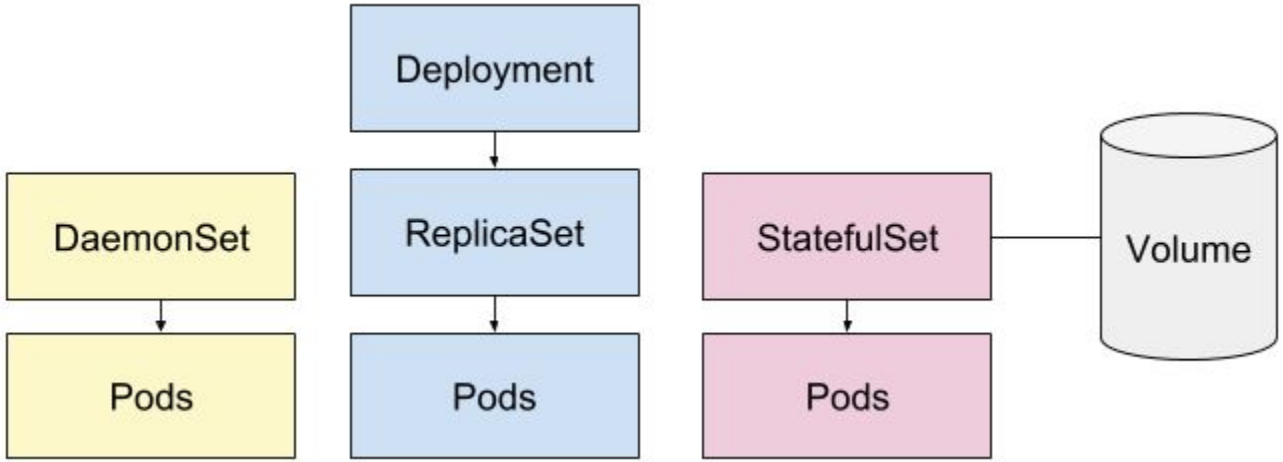
Node Components - Container RT

- Docker
- Containerd
- cri-o
- rktlet

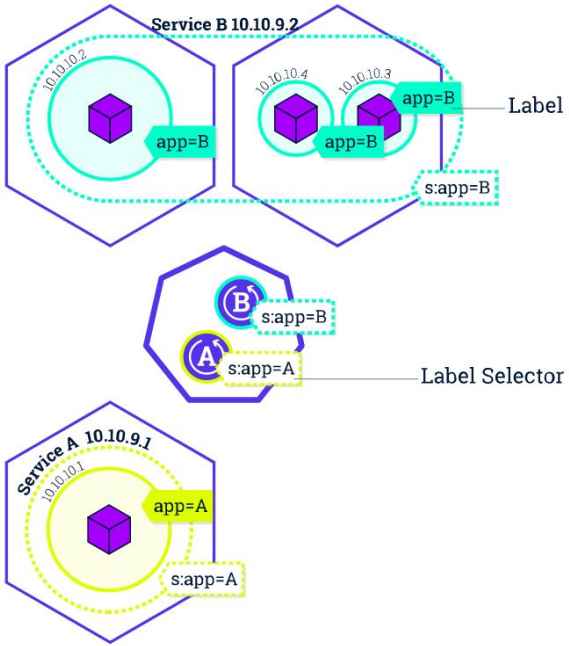
Addon Components

- Cluster DNS
  - Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services
- Web UI
  - General purpose, web-based UI for Kubernetes clusters to view and manager cluster
- Container Resource Monitoring
  - Generic time-series metrics about containers in a central database, and provides a UI for browsing that data
- Cluster level Logging
  - Mechanism responsible for saving container logs to a central log store with search/browsing interface

Objects


Deployment
DaemonSet
ReplicaSet
StatefulSet
Volume
Pods
Pods
Pods

Objects


Service B 10.10.9.2
10.10.10.2
app=B
10.10.10.4
10.10.10.3
app=B
app=B
Label
s:app=B
B
s:app=B
A
s:app=A
Label Selector
Service A 10.10.9.1
10.10.10.1
app=A
s:app=A

Module 2: Hello minikube

- Tools
  - kubectl
  - minikube
- Local Cluster
- Exercises

Tools - minikube

- Tool that makes k8s run locally
- Creates single node cluster inside VM for dev purpose

Features:

- DNS
- NodePorts
- ConfigMaps & Secrets
- Dashboards
- Container Runtime
- Container Networking
- Ingress

Tools Installation

Kubectl

- `curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl`
- `chmod +x ./kubectl`
- `sudo mv ./kubectl /usr/local/bin/kubectl`

Minikube

- `curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube`
- `sudo mkdir -p /usr/local/bin/`
- `sudo install minikube /usr/local/bin/`

NOTE: Ref for windows https://www.studytrails.com/devops/kubernetes/install-minikube-and-docker-with-virtualbox-on-windows-10-home/

Cluster Setup

Cluster startup

`minikube start --cpus 2 --memory 4096 --disk-size 20000 -p rajesh`

TIP

To use minikube's built-in docker env: `eval $(minikube docker-env)`
To login into local cluster: `minikube ssh`

Tools - kubectl

A command line interface for running commands against Kubernetes clusters

Nomenclature

`kubectl [command] [TYPE] [NAME] [flags]`

Where as:

- Command - Operation to be performed on resource ex: create get describe delete run
- TYPE - Type of resource ex: pod, service, deployment
- NAME - Name of resource ex: hello-world, webapp
- Flags - Optional flags ex: --server

Examples

- kubectl run hello --image=tutum/hello-world --port=80
- kubectl run -it busybox --image=busybox --restart=Never
- kubectl run nginx --image=nginx --replicas=1
- ~/git/kubernetes/kubectl-minikube$ kubectl apply -f tutum.yaml

kubectl - commands

```
kubectl get pods
kubectl describe pod hello-world
kubectl describe  pod/nodejs
kubectl delete pod webapp
kubectl get pod -f ./mypods.yaml
kubectl apply -f ./myapp.yaml
kubectl cluster-info
kubectl get pods -o yaml
kubectl get services -o json
kubectl get pods --sort-by=.metadata.name
kubectl get rs,deployments,service
kubectl describe nods
kubectl get pod/<pod-name> svc/<svc-name>
kubectl get pod -l name=<label-name>
kubectl delete pods --all
kubectl get nodes -o json | jq '.items[] | {name:.metadata.name, cap:.status.capacity}'
kubectl get nodes -o yaml | egrep '\sname:|cpu:|memory:'
kubectl get all
```

Exercises (15 mins)

Run Hello World POD using tutum/hello-world image (kubectl run...) & then

- Get POD summary (kubectl get ...)
- Get POD details (kubectl describe ...)
- Get POD IP (kubectl describe pod... -o yaml | egrep....podIP:)
- Login into minikube and verify container is running (using curl)
- Delete the POD created above (kubectl delete ...)
- Verify using kubectl get all
- View cluster info

Module 3: Objects - in detail

- POD
- Controllers
  - Deployment
  - ReplicaSet
  - Service
  - StatefulSet
  - DaemonSet

POD

- Overview
- Lifecycle
- Init Containers
- Preset
- Topology Spread
- Ephemeral Containers

POD - Overview

- ❖ Smallest deployable unit
- ❖ Supports multiple cooperating processes (containers) that form cohesive unit of service
- ❖ Ephemeral Entity

Encapsulates

- application container(s)
- Storage resources
- Unique network IP

Shared Resources:

- Networking
- Storage

Example

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
   app: myapp
spec:
 restartPolicy: Never
 containers:
 - name: myapp-container
   image: busybox:1.28
   command: ['sh', '-c', 'echo The app is running! && sleep 1']
```

Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis:5.0.4
    command:
      - redis-server
      - "/redis-master/redis.conf"
    env:
    - name: MASTER
      value: "true"
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /redis-master-data
      name: data
    - mountPath: /redis-master
      name: config
  volumes:
    - name: data
      emptyDir: {}
    - name: config
      configMap:
        name: example-redis-config
        items:
        - key: redis-config
          path: redis.conf
```

POD - Lifecycle

- Phase
  - Pending
  - Running
  - Succeeded
  - Failed
  - Unknown
- Status
  - lastProbeTime, lastTransitionTime, Message, reason, status, type (PodScheduled, Ready, Initialized, Unscheduleable, ContainerReady)

- Probes
  - Startup, Readiness, Liveness
- Container States
  - Waiting, Running, Terminated
- Readiness Gate (Additional conditions for readiness)
- Restart Policy (Always, Never, OnFailure)
- Lifetime

Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

POD Init Containers

- Always run to completion
- Must complete successfully before next one
- Readiness probes not supported
- Run(s) before application containers

Examples:

- Custom code / utilities to run before app containers
- Block / delay app container startup
- App container image building can be separate

POD Init - Statuses

- Init:N/M
- Init:Error
- Init:CrashLoopBackOff
- Pending
- PodIniliazing
- Running

Examples

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

POD Preset

```yaml
kind: PodPreset
apiVersion: settings/v1alpha1
metadata:
  name: allow-database
  namespace: myns
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: 6379
  volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

POD - Topology Spread

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: zone
            operator: NotIn
            values:
            - zoneC
  containers:
  - name: pause
    image: k8s.gcr.io/pause:3.1
```

POD - Ephemeral

- Meant for interactive troubleshooting inside POD
- No resource guarantees
- Never restart automatically
- Process Namespace sharing

```
kubectl replace --raw /api/v1/namespaces/default/pods/example-pod/ephemeralcontainers -f ec.json
```

Controllers - ReplicaSet

To maintain stable set of replica PODs running at any given time

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Controllers - Deployment

Use Cases

- To rollout a ReplicaSet
- To declare a new set of PODs
- To rollback to an earlier version of deployment
- To scale up deployment to facilitate more load
- To pause the deployment / rollout

Deployment - Example

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Controllers - StatefulSet

Use Cases

- Stable, unique network identifiers
- Stable, persistent storage
- Ordered, graceful deployment and scaling
- Ordered, automated rolling updates

Limitations

- Requires headless service (manual way)
- No automatic deletion of referenced volumes
- No PODs deletion guarantee when StatefulSet is deleted
- Rolling Updates not consistent always

StatefulSet - Example

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Controllers - DaemonSet

Purpose

- To run a copy of a POD on all / some node(s)

Use Cases

- Storage cluster daemon (gluster, ceph)
- Log Collectors (fluentd, logstash)
- Node Monitoring daemons (Prometheus, Dynatrace, collectd)

DaemonSet - Example

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Exercises (30 mins)

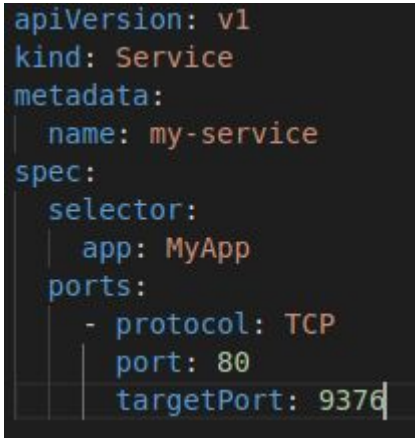
- Create POD with init containers
  - Main app container from tutum/hello-world
  - Init container using busybox to fetch google.com/index.html and save to /www/google.html
- Create a deployment to run 3 replicas of nginx container
- Scale down the replicas to 1
- Scale up replicas to 8
- View the roll out history
- Switch to rollout version 2
- View deployments, rc, pod using kubectl
- Scale down replicas to 5
- Update image to nginx:1.9.1 and immediately try another rollout with nginx:1.7.1
- Observe if rollout with nginx:1.9.1 was completed or not

Module 4: Services & Load Balancing

- Service
- Topology
- DNS
- Ingress

Service

- An abstract way to expose an application running on pod as network service.
- Frontends and backends of application can connect without worrying about POD IPs



Service without selector

Use Cases:

- External DB Cluster in production
- To point a service in another namespace / cluster
- Uses session affinity while connecting to backend PODs

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

Service Types

- Cluster IP
  - Service exposed on cluster internal IP
  - Reachable only within cluster
- Node Port
  - Exposed on each Node IP at static port
- Load Balancer
  - Exposed through external cloud load balancer
- External Name
  - Exposed through the contents of external field via CNAME record

External IP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
  externalIPs:
    - 80.11.12.10
```

DNS

DNS Policy

- Default
  - Inherit name resolution from Node
- ClusterFirst
  - Forwards to upstream nameserver for unresolved name queries
- ClusterFirstWithHostNet
  - Only for PODs running with hostNetwork
- None
  - POD explicitly defines it using dnsConfig

DNS - Example

```yaml
apiVersion: v1
kind: Service
metadata:
  name: default-subdomain
spec:
  selector:
    name: busybox
  clusterIP: None
  ports:
  - name: foo # Actually, no port is needed.
    port: 1234
    targetPort: 1234
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox1
  labels:
    name: busybox
spec:
  hostname: busybox-1
  subdomain: default-subdomain
  containers:
  - image: busybox:1.28
    command:
      - sleep
      - "3600"
    name: busybox
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox2
  labels:
    name: busybox
spec:
  hostname: busybox-2
  subdomain: default-subdomain
  containers:
  - image: busybox:1.28
    command:
      - sleep
      - "3600"
    name: busybox
```

Custom DNS - Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 1.2.3.4
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Ingress

- Provides load balancing, SSL Termination and Name based virtual hosting
- Provides externally reachable URLs to Services
- Used for HTTP / HTTPS protocols

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

Exercises (30 mins)

- Create deployment based on nginx image with 3 replicas
- Create NodePort service to map to the PODs created by above deployment
- View service URL and access it using browser
- Create another service of type Ingress
- View service URL and Access the service outside cluster
- Create one more service of type ClusterIP
- View service URL and find a way to access it

Module 5: Storage

- Volumes
- Persistent Volumes
- Claims
- Static & Dynamic
- Lab Exercises

Overview

- Ephemeral
  - Tightly couple with POD lifetime
  - Deleted when POD is removed
  - Example: emptydir, hostpath

- Persistent
  - Survives POD reboots
  - Meant for long term and independent of POD / Node lifecycle
  - Examples: NFS, Cloud storage (EBS etc)

Examples - emptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

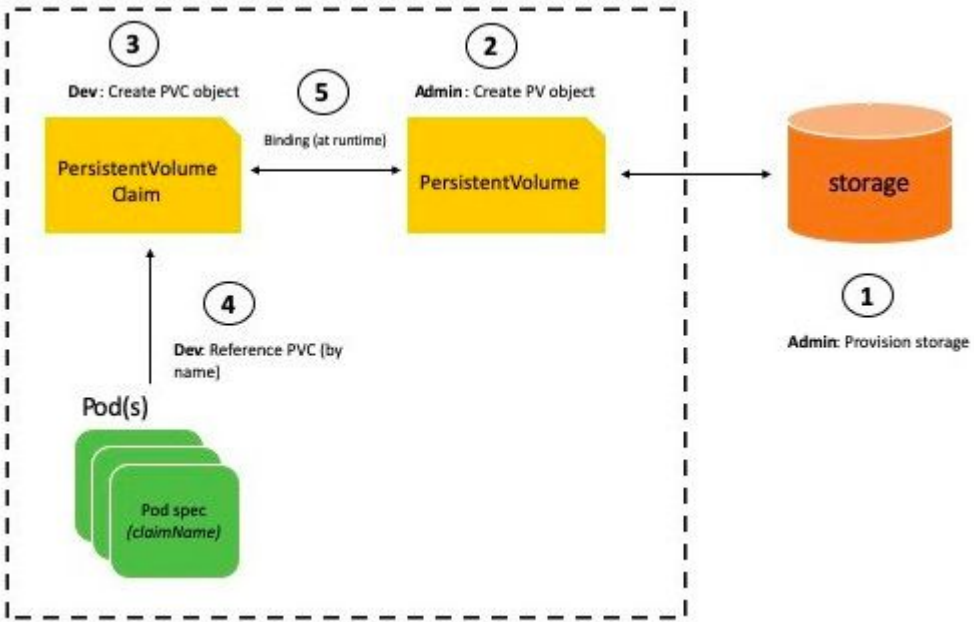
Examples - hostpath (file/dir)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: DirectoryOrCreate
```

Persistent Volume - static & local

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  # volumeMode field requires BlockVolume Alpha feature gate to be enabled.
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

Persistent Volume - static



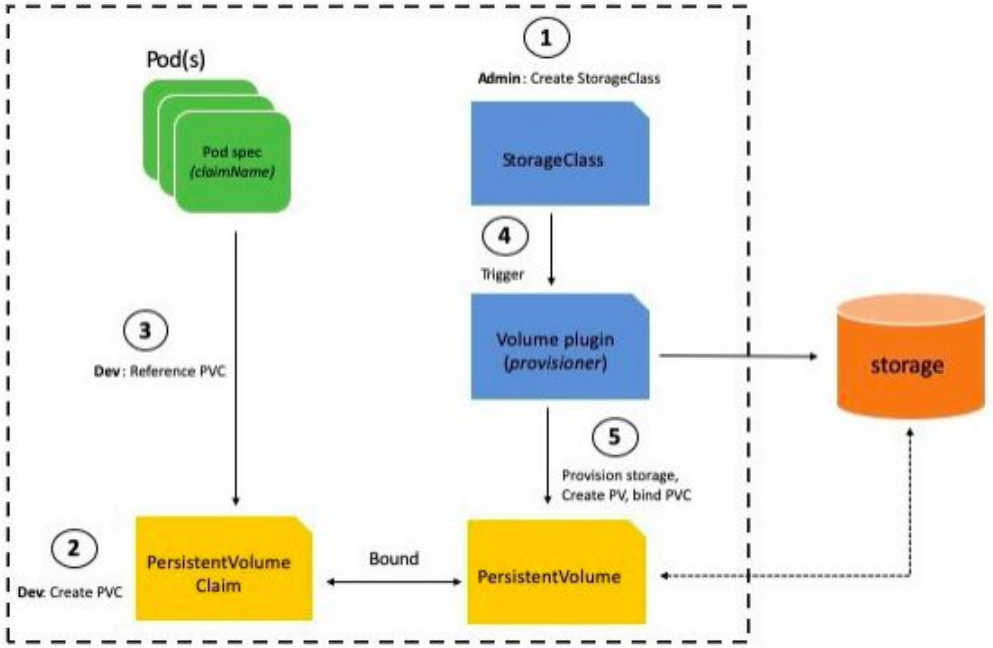
Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Persistent Volume - Dynamic



Persistent Volumes

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- AzureDisk
- CSI
- FC (Fibre Channel)
- FlexVolume
- Flocker
- NFS
- iSCSI
- RBD (Ceph Block Device)
- CephFS
- Cinder (OpenStack block storage)
- Glusterfs
- VsphereVolume
- Quobyte Volumes
- HostPath (Single node testing only – local storage is not supported in any way and WILL NOT WORK in a multi-node cluster)
- Portworx Volumes
- ScaleIO Volumes
- StorageOS

Exercises (15 mins)

- Create POD (nginx / redis) to use volume emptyDir
- Launch POD and login into POD
- Create test file
- Kill the container process (nginx / redis)
- Observe POD status and login into POD again
- Verify if test file exists

  NOTE:

  POD has restartPolicy as Always

  Ephemeral storage is associated till POD is deleted

Exercises (15 mins)

- Create nginx POD that uses pvc for serving web files
- Define pvc that uses pv
- Define pv that refers to host path /mnt/data
- Create index.html echoing ‘hello k8s’ under host path
- Verify that nginx serves the index.html contents that you saved

Module 6: Configuration

- Best Practices
- Resource Bin Packing
- Managing compute Resources
- Secrets
- Cluster Access
- Security (4 Cs)

Best Practices

- Configuration - specify latest stable API version
- Keep config files in version control before pushing to cluster
- Prefer YAML over JSON
- Group related objects into one file whenever it makes more sense
- Don’t specify default values unnecessarily
- Put Object descriptions as part of annotations
- Don’t use naked PODs
- Create service before deployments
- Avoid using hostPort for POD
- Use labels effectively
- Use image tag instead of using latest as the default
- Use kubectl run and expose to launch single container deployments & services

Module 7: POD - Examples

- Memory & CPU Resources
- Volume, Persistence Volume
- Liveness, Ready-ness and startup probes
- Shared namespaces among containers
- Static POD
- Assign POD to Nodes
- Command & Arguments
- Env Variables
- Expose POD info to containers
- PODPreset

Module 8: Applications & Jobs

- Stateless
- Stateful
- Rolling updates
- Scaling & Autoscaling
- Cron Job
- Parallel Processing

Module 9: Debugging, Logging

- Introspection & Debugging
- Auditing
- Debug - Statefulset
- Debug - Init Containers
- Debug - POD, RC and Services
- Debug - live debugging with Telepresence
- Stackdriver events
- Shell to running container
- Troubleshooting - clusters

Commands - Introspection & debugging

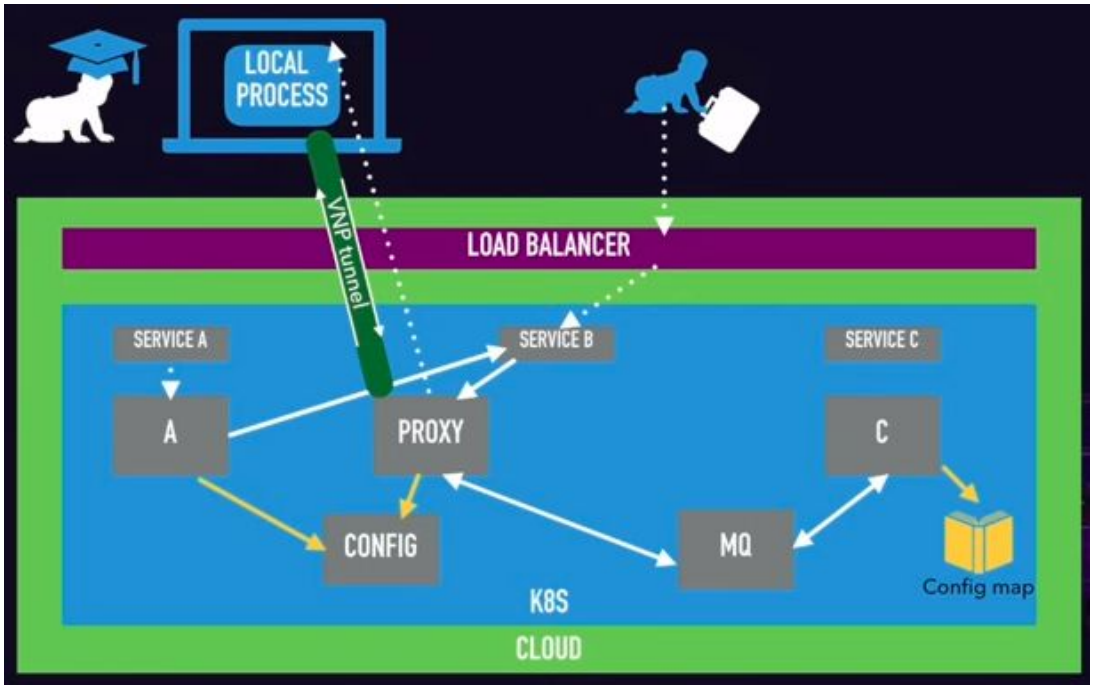
kubectl get pods
kubectl get pod <pod-name> -o yaml
kubectl describe <pod-name>
kubectl describe <pod-name> -o yaml
kubectl get events
kubectl get events --namespace=my-namespace (--all-namespaces)
kubectl get nodes
kubectl get node <node-name>
kubectl get node <node-name> -o yaml
kubectl describe node <node-name>
kubectl describe node <node-name> -o yaml

Live debugging using IDE

telepresence --swap-deployment hostnames --namespace default --run mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005"

Project: git/rest-service

Telepresence - Live debugging



Shell to running container

rajesh@rajesh-Gazelle:~/git/kubernetes/debugging/shell$ kubectl apply -f shell-demo.yaml

kubectl get pod shell-demo
kubectl exec -it shell-demo -- /bin/bash

root@shell-demo:/# ls /

root@shell-demo:/# echo Hello shell demo > /usr/share/nginx/html/index.html
root@shell-demo:/# apt-get update
root@shell-demo:/# apt-get install curl
root@shell-demo:/# curl localhost

kubectl exec shell-demo env

kubectl exec -it my-pod --container main-app -- /bin/bash

Troubleshooting - Cluster

- Listing nodes
- Looking at logs
- Root Causes
- Possible Scenarios
- Mitigations

Module 10: Accessing App via Cluster

- Web UI
- Accessing clusters
- Port forwarding
- Service and Load Balancer
- Ingress on Minikube
- Share volume across containers
- Configure DNS