**Minikube and Kubernetes Setup Guide**

**1. Minikube Installation**

Download and install Minikube using the following commands:

bash

Copy code

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube\_latest\_amd64.deb

sudo dpkg -i minikube\_latest\_amd64.deb

**2. Kubectl Installation**

Download and install kubectl with:

bash

Copy code

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

**Minikube Cluster Management**

**Start Minikube Cluster**

Start a Minikube cluster with two nodes:

bash

Copy code

minikube start --nodes 2 -p local-cluster --driver=docker --force

minikube status -p local-cluster

**Check Cluster Nodes and Contexts**

bash

Copy code

kubectl get nodes

kubectl get pods

kubectl config get-contexts

**Add or Delete Minikube Nodes**

* **Add a worker node:**

bash

Copy code

minikube node add --worker -p local-cluster

* **Delete a specific node:**

bash

Copy code

minikube node delete local-cluster-m03 -p local-cluster

**Access Minikube Dashboard**

bash

Copy code

minikube dashboard --url -p local-cluster

**Working with Pods**

**Sample Pod Configuration (nginx)**

**nginx-pod.yaml**

yaml

Copy code

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod1

labels:

team: development

app: nginx1

spec:

containers:

- name: nginx-container

image: nginx:latest

ports:

- containerPort: 80

**Basic Pod Commands**

* **Run an Nginx Pod:**

bash

Copy code

kubectl run nginx-pod --image=nginx

* **Get Pods with a Specific Label:**

bash

Copy code

kubectl get pods -l team=integration

* **Get Pod Details (wide output):**

bash

Copy code

kubectl get pod nginx-pod1 -o wide

**Interacting with Pods**

* **Access Pod Shell:**

bash

Copy code

kubectl exec -it nginx-pod1 -- bash

* **Port Forwarding:**

bash

Copy code

kubectl port-forward nginx-pod1 8083:80

* **View Pod Logs:**

bash

Copy code

kubectl logs nginx-pod1

**Deleting Pods**

* **Delete Pod Using YAML File:**

bash

Copy code

kubectl delete -f nginx-pod.yaml

* **Delete Pod Without YAML File:**

bash

Copy code

kubectl delete pod nginx-pod

**ReplicaSets and Deployments**

**Checking ReplicaSet Resources**

bash

Copy code

kubectl api-resources | grep replicaset

**Sample ReplicaSet Configuration**

**postgresq-replicaset.yaml**

yaml

Copy code

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: postgresq-replicaset

spec:

replicas: 3

selector:

matchLabels:

app: postgresql

template:

metadata:

name: postgresql-pod

labels:

app: postgresql

spec:

containers:

- name: postgresql-container

image: postgres:latest

ports:

- containerPort: 5432

env:

- name: POSTGRES\_PASSWORD

value: P@ssw0rd

**List ReplicaSets**

bash

Copy code

kubectl get rs

**Rollouts and Rollbacks**

* **Delete All Resources:**

bash

Copy code

kubectl delete all --all

* **Show Pod Labels:**

bash

Copy code

kubectl get po --show-labels

**Sample Deployment Configuration**

**postgresq-deployment.yaml**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: postgresq-deployment

spec:

replicas: 3

selector:

matchLabels:

app: postgresql

template:

metadata:

name: postgresql-pod

labels:

app: postgresql

spec:

containers:

- name: postgresql-container

image: postgres:latest

ports:

- containerPort: 5432

env:

- name: POSTGRES\_PASSWORD

value: P@ssw0rd

**Scaling Deployments**

bash

Copy code

kubectl scale --replicas=4 deployment/nginx-deployment

**Viewing Rollout History**

bash

Copy code

kubectl rollout history deployment/nginx-deployment

**Adding Annotations for Change Cause**

**Updated Deployment with Annotation**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: postgresq-deployment

annotations:

kubernetes.io/change-cause: "update the version from latest to 1.26 for downgrade the application"

spec:

replicas: 3

selector:

matchLabels:

app: postgresql

template:

metadata:

name: postgresql-pod

labels:

app: postgresql

spec:

containers:

- name: postgresql-container

image: postgres:15.9

ports:

- containerPort: 5432

env:

- name: POSTGRES\_PASSWORD

value: P@ssw0rd

**Rolling Back to a Previous Version**

bash

Copy code

kubectl rollout history deployment/nginx-deployment

kubectl rollout undo deployment/nginx-deployment --to-revision=5

kubectl rollout status deployment/nginx-deployment

**Services**

**Command to Get Endpoints**

bash

Copy code

kubectl get endpoints

**Service Configuration**

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: NodePort

selector:

app: nginx

ports:

- port: 8082

targetPort: 80

nodePort: 30000

**Ingress Configuration**

**Path-Based Routing**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: nginx-ingress-path-based

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /$1

spec:

rules:

- host: nginx-demo.com

http:

paths:

- path: /(.\*)

pathType: Prefix

backend:

service:

name: nginx-service

port:

number: 8082

- path: /api/(.\*)

pathType: Prefix

backend:

service:

name: nginx-api-service

port:

number: 3001

**Host-Based Routing**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: nginx-ingress-host-based

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /$1

spec:

rules:

- host: nginx-demo.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: nginx-service

port:

number: 8082

- host: nginx-api-demo.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: nginx-api-service

port:

number: 3001

**Namespace Management**

**Predefined Namespaces**

1. **Default**: Resources without explicit namespace assignment.
2. **Kube-node-lease**: Heartbeat leases for nodes.
3. **Kube-public**: Public resources.
4. **Kube-system**: Kubernetes system objects.

**Creating Namespaces**

**Command**

bash

Copy code

kubectl create namespace developer

kubectl create namespace staging

kubectl create namespace prod

**YAML**

apiVersion: v1

kind: Namespace

metadata:

name: staging

**Volume Management**

**Temporary Data Maintenance**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

annotations:

kubernetes.io/change-cause: "Update version to 1.26"

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx-container

image: nginx:1.26

ports:

- containerPort: 80

volumeMounts:

- mountPath: /usr/share/nginx/html

name: nginx-volume

volumes:

- name: nginx-volume

emptyDir: {}

**Persistent Volume**

yaml

Copy code

apiVersion: v1

kind: PersistentVolume

metadata:

name: nginx-pv

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: /mnt/data

persistentVolumeReclaimPolicy: Retain

**Persistent Volume Claim**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: nginx-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

**Application Deployments**

**Deployment Example**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment1

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx-container

image: nginx:latest

ports:

- containerPort: 80

volumeMounts:

- mountPath: /usr/share/nginx/html

name: nginx-volume

volumes:

- name: nginx-volume

persistentVolumeClaim:

claimName: nginx-pvc

**StatefulSet Example**

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: postgresql-statefulset

spec:

replicas: 3

selector:

matchLabels:

app: postgresql

template:

metadata:

labels:

app: postgresql

spec:

containers:

- name: postgresql-container

image: postgres:15.9

ports:

- containerPort: 5432

env:

- name: POSTGRES\_PASSWORD

value: P@ssw0rd

volumeMounts:

- mountPath: /var/lib/postgresql/data

name: postgres-data-volume

volumeClaimTemplates:

- metadata:

name: postgres-data-volume

spec:

accessModes: ["ReadWriteOnce"]

resources:

requests:

storage: 1Gi

**Resource Management**

**Resource Configuration**

resources:

requests:

cpu: "1"

memory: "1Gi"

limits:

cpu: "2"

memory: "2Gi"

**LimitRange**

apiVersion: v1

kind: LimitRange

metadata:

name: limit-range

spec:

limits:

- type: Pod

min:

cpu: 50m

memory: 5Mi

- type: Container

defaultRequest:

cpu: 100m

memory: 10Mi

default:

cpu: 200m

memory: 100Mi

min:

cpu: 50m

memory: 5Mi

max:

cpu: "1"

memory: 1Gi

maxLimitRequestRatio:

cpu: "2"

memory: "4"

**ResourceQuota**

apiVersion: v1

kind: ResourceQuota

metadata:

name: resource-quota

spec:

hard:

requests.cpu: "2"

requests.memory: 1Gi

limits.cpu: "4"

limits.memory: 2Gi

**Advanced Scheduling in Kubernetes**

**1. nodeName** The nodeName field in the Pod specification is used to manually bind a Pod to a specific node, overriding Kubernetes' default scheduler behavior. By specifying nodeName, the Pod will be scheduled only on the defined node.

**Example:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

annotations:

kubernetes.io/change-cause: "update the version from latest to 1.26 for downgrade the application"

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

name: nginx-pod

labels:

app: nginx

spec:

nodeName: local-cluster-m02

containers:

- name: nginx-container

image: nginx:1.26

ports:

- containerPort: 80

**2. nodeSelector** nodeSelector allows you to constrain a Pod to run on nodes that have specific labels. It is a simpler method for node selection based on labels, ensuring that Kubernetes schedules the Pod on only those nodes matching the label(s).

**Label the nodes:**

kubectl label node <node\_name> team=development

kubectl label node local-cluster-m03 team=qa

kubectl label node local-cluster team=staging

kubectl label node local-cluster-m02 team=production

**Example:**

spec:

nodeSelector:

team: qa

**3. Node Affinity** Node affinity is a more flexible way to control on which nodes Pods can be scheduled. It allows more advanced rules compared to nodeSelector, supporting both hard and soft affinity rules.

* **Hard Rules**: Required during scheduling (requiredDuringSchedulingIgnoredDuringExecution).
* **Soft Rules**: Preferred during scheduling (preferredDuringSchedulingIgnoredDuringExecution).

**Example - Hard Node Affinity:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

annotations:

kubernetes.io/change-cause: "update the version from latest to 1.26 for downgrade the application"

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

name: nginx-pod

labels:

app: nginx

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: team

operator: In

values:

- qa

containers:

- name: nginx-container

image: nginx:1.26

ports:

- containerPort: 80

**Example - Soft Node Affinity:**

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 30

preference:

matchExpressions:

- key: team

operator: In

values:

- qa

- weight: 70

preference:

matchExpressions:

- key: team

operator: In

values:

- staging

containers:

- name: nginx

image: nginx

imagePullPolicy: IfNotPresent

**Autoscaling in Kubernetes**

**Types of Autoscalers in Kubernetes:**

1. **Horizontal Pod Autoscaler (HPA)**
2. **Vertical Pod Autoscaler (VPA)**
3. **Cluster Autoscaler**

**1. Horizontal Pod Autoscaler (HPA)**

**What it does:**  
The HPA automatically adjusts the number of replicas of a Pod in a Deployment, ReplicaSet, or StatefulSet based on CPU, memory usage, or custom metrics.

**Key Features:**

* Monitors CPU, memory, or custom metrics via the Metrics Server.
* Scales Pods up or down based on workload demand.
* Supports custom metrics, like queue length or request latency.

**Example:**

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: hpa-example

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

**2. Vertical Pod Autoscaler (VPA)**

**What it does:**  
The VPA adjusts the resource requests and limits (CPU/memory) of Pods to ensure optimal resource usage.

**Key Features:**

* Automatically recommends or applies changes to resource requests and limits.
* Operates in recommendation, auto, or manual mode.
* Helps reduce resource wastage and ensures sufficient resources for Pods.

**Example:**

apiVersion: autoscaling.k8s.io/v1

kind: VerticalPodAutoscaler

metadata:

name: vpa-example

spec:

targetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

updatePolicy:

updateMode: "Auto" # Options: "Off", "Auto", "Initial"

**3. Cluster Autoscaler**

**What it does:**  
The Cluster Autoscaler adjusts the number of nodes in a cluster based on resource demands.

**Key Features:**

* Adds nodes when Pods cannot be scheduled due to resource constraints.
* Removes underutilized nodes when Pods can be rescheduled elsewhere.
* Works with cloud providers (e.g., AWS, GCP, Azure) to dynamically manage nodes.

**Example:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: cluster-autoscaler

spec:

containers:

- name: cluster-autoscaler

image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.23.0

command:

- ./cluster-autoscaler

- --cloud-provider=gce

- --nodes=1:10:node-group

**4. Custom Autoscalers**

**What it does:**  
Custom autoscalers allow you to implement custom autoscaling logic based on metrics that are specific to your application, such as queue length or external API requests.

**Key Features:**

* Custom metrics are monitored through custom controllers or CRDs (Custom Resource Definitions).
* Both Pods and nodes can be scaled as needed.
* Custom logic and flexibility in scaling based on the application-specific needs.

**Example:**

* A custom autoscaler can be built using Kubernetes API and operators to handle non-standard metrics like processing queue length or external load factors.

**Summary of Autoscalers in Kubernetes**

| **Autoscaler** | **Purpose** | **Key Features** | **Use Case** |
| --- | --- | --- | --- |
| **Horizontal Pod Autoscaler** | Adjusts replicas based on CPU/memory or custom metrics. | Scales Pods dynamically based on metrics. | Scale applications based on load. |
| **Vertical Pod Autoscaler** | Adjusts resource requests and limits for Pods. | Optimizes resource usage by auto-adjusting CPU/memory requests and limits. | Optimize resource usage for apps. |
| **Cluster Autoscaler** | Adjusts node count based on resource demands. | Adds/removes nodes dynamically based on Pod scheduling needs. | Scale cluster size automatically. |
| **Custom Autoscalers** | Implements custom scaling logic based on specific metrics. | Scales Pods and nodes based on custom metrics. | Flexibility for non-standard scaling needs. |

Kubernetes autoscalers ensure that your cluster's resources are efficiently managed based on workload demands. Whether it's scaling Pods dynamically, adjusting resource requests, or even adding/removing nodes, these tools help maintain application performance while optimizing resource usage.

Jobs and cron jobs:

Create the backup job for database using below yaml file

apiVersion: batch/v1

kind: Job

metadata:

name: postgresql-backup-job

spec:

template:

spec:

containers:

- name: postgresql-backup

image: postgres:15.9

command: ["/bin/sh", "-c"]

args:

- "PGPASSWORD=${POSTGRES\_PASSWORD} pg\_dump -h postgresql-service.default.svc.cluster.local -U postgres -F c -b -v -f /backup/postgresql-backup-$(date +%F\_%T).sql"

env:

- name: POSTGRES\_PASSWORD

value: "P@ssw0rd"

volumeMounts:

- name: backup-storage

mountPath: /backup

restartPolicy: Never

volumes:

- name: backup-storage

persistentVolumeClaim:

claimName: backup1-pvc