**Complete Kubernetes Commands Reference Guide**

**Table of Contents**

1. [Adding Nodes](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#adding-nodes)
2. [Kubernetes Objects](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#kubernetes-objects)
3. [Pods](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#pods)
4. [ReplicaSets](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#replicasets)
5. [Deployments](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#deployments)
6. [Namespaces](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#namespaces)
7. [Services](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#services)
8. [YAML Files](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#yaml-files)
9. [Scheduling](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#scheduling)
10. [Taints and Tolerations](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#taints-and-tolerations)
11. [Resource Limitations](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#resource-limitations)
12. [Static Pods](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#static-pods)
13. [Security](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#security)
14. [Volumes](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#volumes)
15. [Network Modules](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#network-modules)
16. [Ingress](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#ingress)
17. [Deployment Strategies](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#deployment-strategies)
18. [ConfigMaps](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#configmaps)
19. [Secrets](https://claude.ai/chat/f70bd375-5dd1-420d-9697-122b73bd4959#secrets)

**Adding Nodes**

**Node Management Commands**

# View cluster nodes

kubectl get nodes

kubectl get nodes -o wide

# Describe node details

kubectl describe node <node-name>

# Add label to node

kubectl label nodes <node-name> <key>=<value>

kubectl label nodes worker-1 environment=production

# Remove label from node

kubectl label nodes <node-name> <key>-

# Cordon node (mark as unschedulable)

kubectl cordon <node-name>

# Uncordon node (mark as schedulable)

kubectl uncordon <node-name>

# Drain node (evict all pods)

kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data

# View node resource usage

kubectl top nodes

# Get node conditions

kubectl get nodes -o jsonpath='{.items[\*].status.conditions[?(@.type=="Ready")].status}'

# Node join command (on master)

kubeadm token create --print-join-command

# Join worker node to cluster

kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash <hash>

# Reset node

kubeadm reset

**Node Initialization (Master)**

# Initialize cluster

kubeadm init --pod-network-cidr=192.168.0.0/16

# Set up kubectl for current user

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

# Install network plugin (example: Calico)

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

**Kubernetes Objects**

**General Object Commands**

# Get all resources

kubectl get all

kubectl get all -A # All namespaces

# Get specific resource types

kubectl get <resource-type>

kubectl get pods,services,deployments

# Get resources with labels

kubectl get pods -l app=nginx

kubectl get pods --selector=environment=production

# Get resources with field selectors

kubectl get pods --field-selector=status.phase=Running

# Describe any resource

kubectl describe <resource-type> <resource-name>

# Delete resources

kubectl delete <resource-type> <resource-name>

kubectl delete -f <yaml-file>

# Apply configuration

kubectl apply -f <yaml-file>

kubectl apply -f <directory>/

# Create resources imperatively

kubectl create <resource-type> <resource-name> [options]

# Edit resources

kubectl edit <resource-type> <resource-name>

# Get resource definition

kubectl get <resource-type> <resource-name> -o yaml > resource.yaml

# Watch resources

kubectl get pods -w

kubectl get events -w

# Show resource usage

kubectl top pods

kubectl top nodes

**Pods**

**Pod Management**

# List pods

kubectl get pods

kubectl get pods -o wide

kubectl get pods -A # All namespaces

# Create pod from image

kubectl run <pod-name> --image=<image-name>

kubectl run nginx-pod --image=nginx:latest

# Create pod with specific port

kubectl run nginx-pod --image=nginx --port=80

# Execute command in pod

kubectl exec -it <pod-name> -- <command>

kubectl exec -it nginx-pod -- /bin/bash

# Execute command in specific container

kubectl exec -it <pod-name> -c <container-name> -- <command>

# View pod logs

kubectl logs <pod-name>

kubectl logs <pod-name> -c <container-name> # Specific container

kubectl logs <pod-name> -f # Follow logs

kubectl logs <pod-name> --previous # Previous container logs

# Port forwarding

kubectl port-forward <pod-name> <local-port>:<pod-port>

kubectl port-forward nginx-pod 8080:80

# Copy files to/from pod

kubectl cp <local-path> <pod-name>:<pod-path>

kubectl cp <pod-name>:<pod-path> <local-path>

# Delete pod

kubectl delete pod <pod-name>

kubectl delete pod <pod-name> --force --grace-period=0

# Get pod events

kubectl get events --field-selector involvedObject.name=<pod-name>

**Pod YAML Example**

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

environment: production

annotations:

description: "Nginx web server pod"

spec:

containers:

- name: nginx

image: nginx:1.20

ports:

- containerPort: 80

name: http

env:

- name: ENV\_VAR

value: "production"

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

volumeMounts:

- name: config-volume

mountPath: /etc/nginx/conf.d

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 5

periodSeconds: 5

volumes:

- name: config-volume

configMap:

name: nginx-config

restartPolicy: Always

nodeSelector:

environment: production

**ReplicaSets**

**ReplicaSet Management**

# List replicasets

kubectl get replicasets

kubectl get rs

# Create replicaset

kubectl create -f replicaset.yaml

# Scale replicaset

kubectl scale rs <replicaset-name> --replicas=5

# Delete replicaset

kubectl delete rs <replicaset-name>

# Delete replicaset but keep pods

kubectl delete rs <replicaset-name> --cascade=orphan

# Describe replicaset

kubectl describe rs <replicaset-name>

# Edit replicaset

kubectl edit rs <replicaset-name>

**ReplicaSet YAML Example**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-replicaset

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.20

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

**Deployments**

**Deployment Management**

# Create deployment

kubectl create deployment <deployment-name> --image=<image-name>

kubectl create deployment nginx-deploy --image=nginx:1.20

# Create deployment with replicas

kubectl create deployment nginx-deploy --image=nginx:1.20 --replicas=3

# List deployments

kubectl get deployments

kubectl get deploy

# Scale deployment

kubectl scale deployment <deployment-name> --replicas=5

# Update deployment image

kubectl set image deployment/<deployment-name> <container-name>=<new-image>

kubectl set image deployment/nginx-deploy nginx=nginx:1.21

# Rollout status

kubectl rollout status deployment/<deployment-name>

# Rollout history

kubectl rollout history deployment/<deployment-name>

# Rollback deployment

kubectl rollout undo deployment/<deployment-name>

kubectl rollout undo deployment/<deployment-name> --to-revision=2

# Pause/Resume rollout

kubectl rollout pause deployment/<deployment-name>

kubectl rollout resume deployment/<deployment-name>

# Restart deployment

kubectl rollout restart deployment/<deployment-name>

# Autoscale deployment

kubectl autoscale deployment <deployment-name> --min=2 --max=10 --cpu-percent=80

# Delete deployment

kubectl delete deployment <deployment-name>

**Deployment YAML Example**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 1

maxUnavailable: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.20

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 5

periodSeconds: 5

nodeSelector:

environment: production

tolerations:

- key: "key1"

operator: "Equal"

value: "value1"

effect: "NoSchedule"

**Namespaces**

**Namespace Management**

# List namespaces

kubectl get namespaces

kubectl get ns

# Create namespace

kubectl create namespace <namespace-name>

kubectl create ns development

# Delete namespace

kubectl delete namespace <namespace-name>

# Set default namespace

kubectl config set-context --current --namespace=<namespace-name>

# Get resources in specific namespace

kubectl get pods -n <namespace-name>

kubectl get all -n <namespace-name>

# Get resources in all namespaces

kubectl get pods -A

kubectl get all --all-namespaces

# Describe namespace

kubectl describe namespace <namespace-name>

# Create resources in specific namespace

kubectl apply -f resource.yaml -n <namespace-name>

kubectl create deployment nginx --image=nginx -n development

**Namespace YAML Example**

apiVersion: v1

kind: Namespace

metadata:

name: development

labels:

environment: dev

team: backend

annotations:

description: "Development environment namespace"

spec:

finalizers:

- kubernetes

---

# Namespace with resource quotas

apiVersion: v1

kind: ResourceQuota

metadata:

name: dev-quota

namespace: development

spec:

hard:

requests.cpu: "4"

requests.memory: 8Gi

limits.cpu: "8"

limits.memory: 16Gi

pods: "10"

services: "5"

persistentvolumeclaims: "4"

---

# Network policy for namespace

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: deny-all

namespace: development

spec:

podSelector: {}

policyTypes:

- Ingress

- Egress

**Services**

**Service Management**

# List services

kubectl get services

kubectl get svc

# Create service (ClusterIP)

kubectl expose deployment <deployment-name> --port=80 --target-port=8080

# Create NodePort service

kubectl expose deployment <deployment-name> --type=NodePort --port=80

# Create LoadBalancer service

kubectl expose deployment <deployment-name> --type=LoadBalancer --port=80

# Create service imperatively

kubectl create service clusterip <service-name> --tcp=80:8080

# Get service endpoints

kubectl get endpoints <service-name>

# Describe service

kubectl describe service <service-name>

# Delete service

kubectl delete service <service-name>

# Port forward to service

kubectl port-forward service/<service-name> 8080:80

**Service YAML Examples**

**ClusterIP Service**

apiVersion: v1

kind: Service

metadata:

name: nginx-service

labels:

app: nginx

spec:

type: ClusterIP

selector:

app: nginx

ports:

- name: http

port: 80

targetPort: 80

protocol: TCP

**NodePort Service**

apiVersion: v1

kind: Service

metadata:

name: nginx-nodeport

spec:

type: NodePort

selector:

app: nginx

ports:

- port: 80

targetPort: 80

nodePort: 30080

protocol: TCP

**LoadBalancer Service**

apiVersion: v1

kind: Service

metadata:

name: nginx-loadbalancer

spec:

type: LoadBalancer

selector:

app: nginx

ports:

- port: 80

targetPort: 80

protocol: TCP

loadBalancerIP: 192.168.1.100

**Headless Service**

apiVersion: v1

kind: Service

metadata:

name: nginx-headless

spec:

clusterIP: None

selector:

app: nginx

ports:

- port: 80

targetPort: 80

**YAML Files**

**Common YAML Structures**

**Multi-resource YAML**

apiVersion: v1

kind: Namespace

metadata:

name: webapp

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-deployment

namespace: webapp

spec:

replicas: 3

selector:

matchLabels:

app: webapp

template:

metadata:

labels:

app: webapp

spec:

containers:

- name: webapp

image: nginx:1.20

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: webapp-service

namespace: webapp

spec:

selector:

app: webapp

ports:

- port: 80

targetPort: 80

type: ClusterIP

**Complex Pod Specification**

apiVersion: v1

kind: Pod

metadata:

name: complex-pod

labels:

app: webapp

version: v1

annotations:

description: "Complex pod with multiple containers"

spec:

initContainers:

- name: init-myservice

image: busybox:1.28

command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']

containers:

- name: webapp

image: nginx:1.20

ports:

- containerPort: 80

env:

- name: DATABASE\_URL

valueFrom:

secretKeyRef:

name: db-secret

key: url

- name: CONFIG\_PATH

valueFrom:

configMapKeyRef:

name: app-config

key: config.path

volumeMounts:

- name: config-volume

mountPath: /etc/config

- name: data-volume

mountPath: /data

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 80

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /ready

port: 80

initialDelaySeconds: 5

periodSeconds: 5

- name: sidecar

image: busybox:1.28

command: ['sh', '-c', 'while true; do echo sidecar running; sleep 30; done']

volumeMounts:

- name: shared-data

mountPath: /shared

volumes:

- name: config-volume

configMap:

name: app-config

- name: data-volume

persistentVolumeClaim:

claimName: data-pvc

- name: shared-data

emptyDir: {}

nodeSelector:

environment: production

tolerations:

- key: "dedicated"

operator: "Equal"

value: "webapp"

effect: "NoSchedule"

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: environment

operator: In

values: ["production", "staging"]

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

labelSelector:

matchExpressions:

- key: app

operator: In

values: ["webapp"]

topologyKey: kubernetes.io/hostname

**Scheduling**

**Scheduling Commands and Concepts**

# View scheduler events

kubectl get events --sort-by=.metadata.creationTimestamp

# Check pod scheduling

kubectl describe pod <pod-name> | grep -A 10 Events

# Get pods with node information

kubectl get pods -o wide

# Manual scheduling (nodeName in spec)

kubectl apply -f pod-with-nodename.yaml

# Check node capacity and allocation

kubectl describe nodes

# View resource usage

kubectl top nodes

kubectl top pods

# Force reschedule pod

kubectl delete pod <pod-name>

kubectl rollout restart deployment <deployment-name>

**Node Selector Example**

apiVersion: v1

kind: Pod

metadata:

name: scheduled-pod

spec:

nodeSelector:

environment: production

hardware: ssd

containers:

- name: app

image: nginx

**Node Affinity Example**

apiVersion: v1

kind: Pod

metadata:

name: affinity-pod

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: environment

operator: In

values: ["production", "staging"]

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 1

preference:

matchExpressions:

- key: hardware

operator: In

values: ["ssd"]

containers:

- name: app

image: nginx

**Pod Affinity/Anti-Affinity Example**

apiVersion: v1

kind: Pod

metadata:

name: pod-affinity

spec:

affinity:

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values: ["database"]

topologyKey: kubernetes.io/hostname

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

labelSelector:

matchExpressions:

- key: app

operator: In

values: ["web"]

topologyKey: kubernetes.io/hostname

containers:

- name: app

image: nginx

**Taints and Tolerations**

**Taint Management**

# Add taint to node

kubectl taint nodes <node-name> <key>=<value>:<effect>

kubectl taint nodes worker-1 dedicated=webapp:NoSchedule

# Remove taint from node

kubectl taint nodes <node-name> <key>:<effect>-

kubectl taint nodes worker-1 dedicated:NoSchedule-

# List node taints

kubectl describe node <node-name> | grep Taints

# Common taint effects:

# NoSchedule - pods won't be scheduled

# PreferNoSchedule - avoid scheduling if possible

# NoExecute - evict existing pods

# Taint examples

kubectl taint nodes node1 gpu=true:NoSchedule

kubectl taint nodes node2 maintenance=true:NoExecute

kubectl taint nodes node3 special=true:PreferNoSchedule

**Toleration Examples**

apiVersion: v1

kind: Pod

metadata:

name: toleration-pod

spec:

tolerations:

# Exact match toleration

- key: "dedicated"

operator: "Equal"

value: "webapp"

effect: "NoSchedule"

# Exists toleration (any value)

- key: "gpu"

operator: "Exists"

effect: "NoSchedule"

# Tolerate all taints with specific key

- key: "maintenance"

operator: "Exists"

# Tolerate all taints

- operator: "Exists"

# Toleration with tolerationSeconds

- key: "node.kubernetes.io/unreachable"

operator: "Exists"

effect: "NoExecute"

tolerationSeconds: 300

containers:

- name: app

image: nginx

**Deployment with Tolerations**

apiVersion: apps/v1

kind: Deployment

metadata:

name: gpu-deployment

spec:

replicas: 2

selector:

matchLabels:

app: gpu-app

template:

metadata:

labels:

app: gpu-app

spec:

tolerations:

- key: "nvidia.com/gpu"

operator: "Exists"

effect: "NoSchedule"

containers:

- name: gpu-container

image: tensorflow/tensorflow:latest-gpu

resources:

limits:

nvidia.com/gpu: 1

**Resource Limitations**

**Resource Management Commands**

# View resource usage

kubectl top nodes

kubectl top pods

kubectl top pods --containers

# View resource quotas

kubectl get resourcequota

kubectl describe resourcequota <quota-name>

# View limit ranges

kubectl get limitrange

kubectl describe limitrange <limitrange-name>

# Create resource quota

kubectl create quota my-quota --hard=cpu=2,memory=4Gi,pods=10

# Check pod resource requests/limits

kubectl describe pod <pod-name> | grep -A 5 Requests

**Resource Quota Example**

apiVersion: v1

kind: ResourceQuota

metadata:

name: compute-quota

namespace: development

spec:

hard:

# Compute resources

requests.cpu: "4"

requests.memory: 8Gi

limits.cpu: "8"

limits.memory: 16Gi

# Object counts

pods: "10"

services: "5"

secrets: "10"

configmaps: "10"

persistentvolumeclaims: "4"

# Storage

requests.storage: 100Gi

# Quality of service classes

count/pods: "10"

count/services.loadbalancers: "2"

**Limit Range Example**

apiVersion: v1

kind: LimitRange

metadata:

name: resource-limits

namespace: development

spec:

limits:

# Container limits

- type: Container

default:

cpu: "500m"

memory: "512Mi"

defaultRequest:

cpu: "100m"

memory: "128Mi"

max:

cpu: "2"

memory: "2Gi"

min:

cpu: "50m"

memory: "64Mi"

# Pod limits

- type: Pod

max:

cpu: "4"

memory: "4Gi"

min:

cpu: "100m"

memory: "128Mi"

# PVC limits

- type: PersistentVolumeClaim

max:

storage: 10Gi

min:

storage: 1Gi

**Pod with Resource Specifications**

apiVersion: v1

kind: Pod

metadata:

name: resource-demo

spec:

containers:

- name: app

image: nginx

resources:

requests:

memory: "64Mi"

cpu: "250m"

ephemeral-storage: "1Gi"

limits:

memory: "128Mi"

cpu: "500m"

ephemeral-storage: "2Gi"

- name: sidecar

image: busybox

resources:

requests:

memory: "32Mi"

cpu: "100m"

limits:

memory: "64Mi"

cpu: "200m"

**Static Pods**

**Static Pod Management**

# View static pod configuration directory

ls /etc/kubernetes/manifests/

# Check kubelet static pod path

kubectl describe node <node-name> | grep "Static Pod Path"

# Create static pod (place YAML in static pod directory)

sudo cp static-pod.yaml /etc/kubernetes/manifests/

# Remove static pod

sudo rm /etc/kubernetes/manifests/static-pod.yaml

# View static pods (they appear as regular pods)

kubectl get pods -A | grep <node-name>

# Check kubelet logs for static pod errors

journalctl -u kubelet -f

**Static Pod YAML Example**

# File: /etc/kubernetes/manifests/static-web.yaml

apiVersion: v1

kind: Pod

metadata:

name: static-web

labels:

role: myrole

spec:

containers:

- name: web

image: nginx

ports:

- name: web

containerPort: 80

protocol: TCP

volumeMounts:

- name: config

mountPath: /etc/nginx/conf.d

volumes:

- name: config

hostPath:

path: /etc/nginx/conf.d

type: Directory

**Mirror Pod (Automatic for Static Pods)**

Static pods automatically create mirror pods in the API server. They:

* Cannot be controlled via kubectl delete
* Must be removed by deleting the file from static pod directory
* Show the node name as a suffix in the pod name

**Security**

**RBAC (Role-Based Access Control)**

# View current user permissions

kubectl auth can-i <verb> <resource>

kubectl auth can-i create pods

kubectl auth can-i delete deployments --namespace=production

# Check permissions for another user

kubectl auth can-i create pods --as=system:serviceaccount:default:my-sa

# List roles and rolebindings

kubectl get roles

kubectl get rolebindings

kubectl get clusterroles

kubectl get clusterrolebindings

# Create service account

kubectl create serviceaccount <sa-name>

kubectl create sa webapp-sa

# Get service account token

kubectl describe sa <sa-name>

kubectl get secret <token-secret-name> -o yaml

**Service Account Example**

apiVersion: v1

kind: ServiceAccount

metadata:

name: webapp-sa

namespace: default

---

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: webapp-role

namespace: default

rules:

- apiGroups: [""]

resources: ["pods", "services"]

verbs: ["get", "list", "create", "update", "delete"]

- apiGroups: ["apps"]

resources: ["deployments"]

verbs: ["get", "list", "create", "update", "patch"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: webapp-binding

namespace: default

subjects:

- kind: ServiceAccount

name: webapp-sa

namespace: default

roleRef:

kind: Role

name: webapp-role

apiGroup: rbac.authorization.k8s.io

**Security Context Example**

apiVersion: v1

kind: Pod

metadata:

name: security-context-demo

spec:

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: 2000

seLinuxOptions:

level: "s0:c123,c456"

containers:

- name: app

image: nginx

securityContext:

allowPrivilegeEscalation: false

runAsNonRoot: true

runAsUser: 2000

capabilities:

add: ["NET\_ADMIN"]

drop: ["ALL"]

readOnlyRootFilesystem: true

volumeMounts:

- name: tmp-volume

mountPath: /tmp

volumes:

- name: tmp-volume

emptyDir: {}

**Pod Security Policy Example**

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: restricted-psp

spec:

privileged: false

allowPrivilegeEscalation: false

requiredDropCapabilities:

- ALL

volumes:

- 'configMap'

- 'emptyDir'

- 'projected'

- 'secret'

- 'downwardAPI'

- 'persistentVolumeClaim'

runAsUser:

rule: 'MustRunAsNonRoot'

seLinux:

rule: 'RunAsAny'

fsGroup:

rule: 'RunAsAny'

**Network Policy Example**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: webapp-netpol

namespace: default

spec:

podSelector:

matchLabels:

app: webapp

policyTypes:

- Ingress

- Egress

ingress:

- from:

- namespaceSelector:

matchLabels:

name: frontend

- podSelector:

matchLabels:

role: client

ports:

- protocol: TCP

port: 80

egress:

- to:

- namespaceSelector:

matchLabels:

name: database

ports:

- protocol: TCP

port: 5432

**Volumes**

**Volume Management Commands**

# List persistent volumes

kubectl get pv

# List persistent volume claims

kubectl get pvc

# Create PVC

kubectl apply -f pvc.yaml

# Delete PVC

kubectl delete pvc <pvc-name>

# Describe PV/PVC

kubectl describe pv <pv-name>

kubectl describe pvc <pvc-name>

# Check volume usage in pods

kubectl get pods -o custom-columns=NAME:.metadata.name,VOLUMES:.spec.volumes[\*].name

**Persistent Volume Example**

apiVersion: v1

kind: PersistentVolume

metadata:

name: webapp-pv

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

storageClassName: standard

hostPath:

path: /data/webapp

**Persistent Volume Claim Example**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: webapp-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5Gi

storageClassName: standard

**Storage Class Example**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: fast-ssd

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp3

iops: "3000"

throughput: "125"

encrypted: "true"

allowVolumeExpansion: true

reclaimPolicy: Delete

volumeBindingMode: WaitForFirstConsumer

**Pod with Volume Mounts**

apiVersion: v1

kind: Pod

metadata:

name: volume-demo

spec:

containers:

- name: app

image: nginx

volumeMounts:

# Persistent Volume

- name: data-volume

mountPath: /data

# ConfigMap Volume

- name: config-volume

mountPath: /etc/config

# Secret Volume

- name: secret-volume

mountPath: /etc/secrets

readOnly: true

# EmptyDir Volume

- name: cache-volume

mountPath: /cache

# HostPath Volume

- name: host-volume

mountPath: /host-data

volumes:

- name: data-volume

persistentVolumeClaim:

claimName: webapp-pvc

- name: config-volume

configMap:

name: app-config

- name: secret-volume

secret:

secretName: app-secret

- name: cache-volume

emptyDir:

sizeLimit: 1Gi

- name: host-volume

hostPath:

path: /opt/data

type: Directory

**Network Modules**

**Network Commands**

# List network policies

kubectl get networkpolicy

kubectl get netpol

# Describe network policy

kubectl describe networkpolicy <policy-name>

# Test network connectivity

kubectl run test-pod --image=busybox -i --tty --rm -- sh

# Check DNS resolution

kubectl run -i --tty --rm debug --image=busybox --restart=Never -- nslookup kubernetes.default

# View service endpoints

kubectl get endpoints

# Check cluster DNS

kubectl get svc -n kube-system

kubectl logs -n kube-system -l k8s-app=kube-dns

# Network troubleshooting

kubectl exec -it <pod-name> -- netstat -tuln

kubectl exec -it <pod-name> -- ping <target-ip>

**CNI (Container Network Interface) Plugins**

**Calico Installation**

# Install Calico

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

# Check Calico pods

kubectl get pods -n kube-system | grep calico

# Calico network policy example

apiVersion: projectcalico.org/v3

kind: NetworkPolicy

metadata:

name: allow-tcp-6379

namespace: production

spec:

selector: role == 'database'

types:

- Ingress

- Egress

ingress:

- action: Allow

protocol: TCP

source:

selector: role == 'frontend'

destination:

ports:

- 6379

egress:

- action: Allow

**Flannel Installation**

# Install Flannel

kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml

# Check Flannel

kubectl get pods -n kube-system | grep flannel

**Service Mesh (Istio) Example**

# Enable Istio injection

apiVersion: v1

kind: Namespace

metadata:

name: webapp

labels:

istio-injection: enabled

---

# Virtual Service

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: webapp-vs

spec:

http:

- match:

- headers:

version:

exact: v2

route:

- destination:

host: webapp-service

subset: v2

- route:

- destination:

host: webapp-service

subset: v1

weight: 90

- destination:

host: webapp-service

subset: v2

weight: 10

**Ingress**

**Ingress Management Commands**

# List ingress resources

kubectl get ingress

kubectl get ing

# Describe ingress

kubectl describe ingress <ingress-name>

# Create ingress

kubectl create ingress simple-ingress --rule="example.com/path=service:80"

# Delete ingress

kubectl delete ingress <ingress-name>

# Check ingress controller

kubectl get pods -n ingress-nginx

kubectl logs -n ingress-nginx deployment/ingress-nginx-controller

**NGINX Ingress Controller Installation**

# Install NGINX Ingress Controller

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.8.1/deploy/static/provider/cloud/deploy.yaml

# Verify installation

kubectl get pods -n ingress-nginx

kubectl get svc -n ingress-nginx

**Basic Ingress Example**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: webapp-ingress

annotations:

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

spec:

ingressClassName: nginx

rules:

- host: webapp.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: webapp-service

port:

number: 80

**Advanced Ingress Example**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: advanced-ingress

annotations:

# NGINX specific annotations

nginx.ingress.kubernetes.io/ssl-redirect: "true"

nginx.ingress.kubernetes.io/use-regex: "true"

nginx.ingress.kubernetes.io/rewrite-target: /$2

nginx.ingress.kubernetes.io/configuration-snippet: |

more\_set\_headers "X-Frame-Options: SAMEORIGIN";

# Rate limiting

nginx.ingress.kubernetes.io/rate-limit: "100"

nginx.ingress.kubernetes.io/rate-limit-window: "1m"

# Load balancing

nginx.ingress.kubernetes.io/upstream-hash-by: "$request\_uri"

spec:

ingressClassName: nginx

tls:

- hosts:

- webapp.example.com

- api.example.com

secretName: webapp-tls

rules:

- host: webapp.example.com

http:

paths:

- path: /app(/|$)(.\*)

pathType: Prefix

backend:

service:

name: webapp-service

port:

number: 80

- path: /api(/|$)(.\*)

pathType: Prefix

backend:

service:

name: api-service

port:

number: 8080

- host: api.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: api-service

port:

number: 8080

**TLS Certificate Example**

apiVersion: v1

kind: Secret

metadata:

name: webapp-tls

namespace: default

type: kubernetes.io/tls

data:

tls.crt: LS0tLS1CRUdJTi... # base64 encoded certificate

tls.key: LS0tLS1CRUdJTi... # base64 encoded private key

**Deployment Strategies**

**Rolling Update Strategy**

apiVersion: apps/v1

kind: Deployment

metadata:

name: rolling-update-deployment

spec:

replicas: 10

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 2

maxUnavailable: 1

selector:

matchLabels:

app: webapp

template:

metadata:

labels:

app: webapp

spec:

containers:

- name: webapp

image: nginx:1.20

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 5

periodSeconds: 5

**Recreate Strategy**

apiVersion: apps/v1

kind: Deployment

metadata:

name: recreate-deployment

spec:

replicas: 5

strategy:

type: Recreate

selector:

matchLabels:

app: database

template:

metadata:

labels:

app: database

spec:

containers:

- name: database

image: postgres:13

**Blue-Green Deployment**

# Blue version (current)

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-blue

labels:

version: blue

spec:

replicas: 3

selector:

matchLabels:

app: webapp

version: blue

template:

metadata:

labels:

app: webapp

version: blue

spec:

containers:

- name: webapp

image: webapp:v1.0

---

# Green version (new)

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-green

labels:

version: green

spec:

replicas: 3

selector:

matchLabels:

app: webapp

version: green

template:

metadata:

labels:

app: webapp

version: green

spec:

containers:

- name: webapp

image: webapp:v2.0

---

# Service (switch between blue and green)

apiVersion: v1

kind: Service

metadata:

name: webapp-service

spec:

selector:

app: webapp

version: blue # Change to 'green' for cutover

ports:

- port: 80

targetPort: 80

**Canary Deployment Commands**

# Create canary deployment

kubectl create deployment canary-app --image=nginx:1.21 --replicas=1

# Scale main deployment

kubectl scale deployment main-app --replicas=9

# Monitor canary

kubectl get pods -l app=canary-app

kubectl logs -l app=canary-app

# Promote canary

kubectl set image deployment/main-app nginx=nginx:1.21

kubectl delete deployment canary-app

**Canary Deployment Example**

# Main deployment (90% traffic)

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-main

spec:

replicas: 9

selector:

matchLabels:

app: webapp

track: stable

template:

metadata:

labels:

app: webapp

track: stable

spec:

containers:

- name: webapp

image: webapp:v1.0

---

# Canary deployment (10% traffic)

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-canary

spec:

replicas: 1

selector:

matchLabels:

app: webapp

track: canary

template:

metadata:

labels:

app: webapp

track: canary

spec:

containers:

- name: webapp

image: webapp:v2.0

---

# Service for both deployments

apiVersion: v1

kind: Service

metadata:

name: webapp-service

spec:

selector:

app: webapp # Selects both main and canary

ports:

- port: 80

targetPort: 80

**ConfigMaps**

**ConfigMap Management Commands**

# Create ConfigMap from literal values

kubectl create configmap app-config \

--from-literal=database\_url=mongodb://localhost:27017 \

--from-literal=debug=true

# Create ConfigMap from file

kubectl create configmap app-config --from-file=config.properties

# Create ConfigMap from directory

kubectl create configmap app-config --from-file=config-dir/

# Create ConfigMap from env file

kubectl create configmap app-config --from-env-file=.env

# List ConfigMaps

kubectl get configmaps

kubectl get cm

# Describe ConfigMap

kubectl describe configmap app-config

# Edit ConfigMap

kubectl edit configmap app-config

# Delete ConfigMap

kubectl delete configmap app-config

# Get ConfigMap as YAML

kubectl get configmap app-config -o yaml

**ConfigMap YAML Examples**

# Simple ConfigMap

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

namespace: default

data:

database\_url: "mongodb://localhost:27017"

debug: "true"

max\_connections: "100"

config.properties: |

database.host=localhost

database.port=5432

database.name=myapp

log.level=INFO

nginx.conf: |

server {

listen 80;

server\_name localhost;

location / {

root /usr/share/nginx/html;

index index.html;

}

}

---

# Binary data ConfigMap

apiVersion: v1

kind: ConfigMap

metadata:

name: binary-config

binaryData:

config.bin: iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAYAAAAfFcSJAAAADUlEQVR42mP8/5+hHgAHggJ/PchI7wAAAABJRU5ErkJggg==

**Using ConfigMaps in Pods**

apiVersion: v1

kind: Pod

metadata:

name: configmap-demo

spec:

containers:

- name: app

image: nginx

# Environment variables from ConfigMap

env:

- name: DATABASE\_URL

valueFrom:

configMapKeyRef:

name: app-config

key: database\_url

# All keys as environment variables

envFrom:

- configMapRef:

name: app-config

# Volume mounts from ConfigMap

volumeMounts:

- name: config-volume

mountPath: /etc/config

- name: nginx-config

mountPath: /etc/nginx/nginx.conf

subPath: nginx.conf

volumes:

- name: config-volume

configMap:

name: app-config

- name: nginx-config

configMap:

name: app-config

items:

- key: nginx.conf

path: nginx.conf

mode: 0644

**Secrets**

**Secret Management Commands**

# Create secret from literal values

kubectl create secret generic app-secret \

--from-literal=username=admin \

--from-literal=password=secretpassword

# Create secret from files

kubectl create secret generic app-secret --from-file=username.txt --from-file=password.txt

# Create TLS secret

kubectl create secret tls webapp-tls \

--cert=cert.pem \

--key=key.pem

# Create Docker registry secret

kubectl create secret docker-registry regcred \

--docker-server=myregistry.com \

--docker-username=myuser \

--docker-password=mypassword \

--docker-email=myemail@example.com

# List secrets

kubectl get secrets

# Describe secret (doesn't show values)

kubectl describe secret app-secret

# Get secret values (base64 encoded)

kubectl get secret app-secret -o yaml

# Decode secret values

kubectl get secret app-secret -o jsonpath='{.data.password}' | base64 --decode

# Edit secret

kubectl edit secret app-secret

# Delete secret

kubectl delete secret app-secret

**Secret YAML Examples**

# Generic Secret

apiVersion: v1

kind: Secret

metadata:

name: app-secret

type: Opaque

data:

username: YWRtaW4= # base64 encoded 'admin'

password: c2VjcmV0cGFzc3dvcmQ= # base64 encoded 'secretpassword'

---

# TLS Secret

apiVersion: v1

kind: Secret

metadata:

name: webapp-tls

type: kubernetes.io/tls

data:

tls.crt: LS0tLS1CRUdJTi4uLg== # base64 encoded certificate

tls.key: LS0tLS1CRUdJTi4uLg== # base64 encoded private key

---

# Docker Registry Secret

apiVersion: v1

kind: Secret

metadata:

name: regcred

type: kubernetes.io/dockerconfigjson

data:

.dockerconfigjson: eyJhdXRocyI6eyJteXJlZ2lzdHJ5LmNvbSI6eyJ1c2VybmFtZSI6Im15dXNlciIsInBhc3N3b3JkIjoibXlwYXNzd29yZCIsImVtYWlsIjoibXllbWFpbEBleGFtcGxlLmNvbSIsImF1dGgiOiJiWGwxYzJWeU9tMTVjR0Z6YzNkdmNtUT0ifX19

---

# Service Account Token Secret

apiVersion: v1

kind: Secret

metadata:

name: sa-token-secret

annotations:

kubernetes.io/service-account.name: webapp-sa

type: kubernetes.io/service-account-token

**Using Secrets in Pods**

apiVersion: v1

kind: Pod

metadata:

name: secret-demo

spec:

containers:

- name: app

image: nginx

# Environment variables from Secret

env:

- name: DB\_USERNAME

valueFrom:

secretKeyRef:

name: app-secret

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: app-secret

key: password

# All keys as environment variables

envFrom:

- secretRef:

name: app-secret

# Volume mounts from Secret

volumeMounts:

- name: secret-volume

mountPath: /etc/secrets

readOnly: true

- name: tls-volume

mountPath: /etc/tls

readOnly: true

# Image pull secret

imagePullSecrets:

- name: regcred

volumes:

- name: secret-volume

secret:

secretName: app-secret

defaultMode: 0400

- name: tls-volume

secret:

secretName: webapp-tls

items:

- key: tls.crt

path: cert.pem

- key: tls.key

path: key.pem

mode: 0600

**Secret Best Practices**

# Sealed Secrets (Third-party solution)

apiVersion: bitnami.com/v1alpha1

kind: SealedSecret

metadata:

name: app-sealed-secret

spec:

encryptedData:

username: AgBy3i4OJSWK+PiTySYZZA9rO43cGDEQAx...

password: AgBy3i4OJSWK+PiTySYZZA9rO43cGDEQAx...

template:

metadata:

name: app-secret

type: Opaque

---

# External Secrets Operator

apiVersion: external-secrets.io/v1beta1

kind: SecretStore

metadata:

name: vault-backend

spec:

provider:

vault:

server: "https://vault.example.com"

path: "secret"

version: "v2"

auth:

kubernetes:

mountPath: "kubernetes"

role: "demo"

---

apiVersion: external-secrets.io/v1beta1

kind: ExternalSecret

metadata:

name: vault-secret

spec:

refreshInterval: 30s

secretStoreRef:

name: vault-backend

kind: SecretStore

target:

name: app-secret

creationPolicy: Owner

data:

- secretKey: username

remoteRef:

key: database

property: username

- secretKey: password

remoteRef:

key: database

property: password

**Additional Commands and Tips**

**Debugging Commands**

# Get all resources in namespace

kubectl get all -n <namespace>

# Watch resources

kubectl get pods -w

kubectl get events -w

# Resource usage

kubectl top pods --sort-by=cpu

kubectl top nodes --sort-by=cpu

# Cluster information

kubectl cluster-info

kubectl cluster-info dump

# API resources

kubectl api-resources

kubectl api-versions

# Explain resource fields

kubectl explain pod.spec.containers

kubectl explain deployment.spec.strategy

# Dry run

kubectl apply -f deployment.yaml --dry-run=client -o yaml

kubectl create deployment test --image=nginx --dry-run=server -o yaml

# Generate YAML

kubectl create deployment nginx --image=nginx --dry-run=client -o yaml > deployment.yaml

kubectl expose deployment nginx --port=80 --dry-run=client -o yaml > service.yaml

**Useful Aliases**

# Add to ~/.bashrc or ~/.zshrc

alias k='kubectl'

alias kgp='kubectl get pods'

alias kgs='kubectl get svc'

alias kgd='kubectl get deployment'

alias kdp='kubectl describe pod'

alias kdd='kubectl describe deployment'

alias kaf='kubectl apply -f'

alias kdel='kubectl delete'

alias kex='kubectl exec -it'

alias klog='kubectl logs'

alias kpf='kubectl port-forward'

# Function for quick pod shell access

ksh() {

kubectl exec -it $1 -- /bin/bash

}

This comprehensive Kubernetes reference guide covers all the areas you requested with practical examples, commands, and YAML configurations. Each section provides both imperative commands and declarative YAML examples to help you understand and implement Kubernetes effectively.