```
# Basic shortcuts
alias k=kubectl
alias kn='kubectl get nodes'
alias kp='kubectl get pods'
alias kd='kubectl get deployments'
alias ks='kubectl get svc'

# Output formats
alias ky='kubectl get -o yaml'
alias kw='kubectl get -o wide'

# Namespaces
alias kns='kubectl config set-context --current --namespace'
alias ka='kubectl get all --all-namespaces'

# Create with YAML output
export do='--dry-run=client -o yaml'

alias ktmp='kubectl run tmp-${RANDOM} --image=busybox --restart=Never --rm -it -- /bin/sh'
```

source ~/k8s-aliases.sh

Vim:

```
echo 'set autoindent smartindent ts=2 sw=1 et>> ~/.vimrc
```

Vim tips:-
Move to start 0
Move to end $
Indent right >>
Indent left <<
Indent by amount 1> (only in insert mode)
Search /
Jump to next searched word : n
Jump to previous searched word : N

Container Images

Building Images

bash

```
# Build image from Dockerfile in current directory
```

```
docker build -t myapp:1.0 .

# Build with build args
docker build --build-arg VAR=value -t myapp:1.0 .

# Multi-stage build (Dockerfile example)
FROM node:14 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
```

Image Management

bash

```
# Tag and push
docker tag myapp:1.0 registry.example.com/myapp:1.0
docker push registry.example.com/myapp:1.0

# Pull image
docker pull nginx:1.19

# List images
docker images
```

Building & Saving Images as OCI Format

#building image
podman build -t myimage:latest .

#exporting image in oci format
podman save myimage:latest --format oci-archive -o myimage.tar

Best Practices

- Use specific version tags, not `latest`
- Multi-stage builds for smaller images
- Set non-root user: `USER 1000`
- Use `.dockerignore` file
- Minimize layers with && chaining

- Include health checks: `HEALTHCHECK CMD curl --fail http://localhost:8080 || exit 1`

Workload Resources

Deployment

bash

```
# Create deployment
kubectl create deployment nginx --image=nginx:1.19 --replicas=3

# Scale deployment
kubectl scale deployment nginx --replicas=5

# Update image
kubectl set image deployment/nginx nginx=nginx:1.20

# Rollout commands
kubectl rollout status deployment/nginx
kubectl rollout history deployment/nginx
kubectl rollout undo deployment/nginx [--to-revision=2]

kubectl rollout pause/resume deployment/nginx
```

DaemonSet

bash

```
# Create DaemonSet
kubectl create -f daemonset.yaml

# Check DaemonSet
kubectl get daemonset
kubectl describe ds fluentd

# Delete DaemonSet

kubectl delete ds fluentd
```

StatefulSet

bash

```
# Create StatefulSet
kubectl apply -f statefulset.yaml

# Scale StatefulSet
kubectl scale sts web --replicas=5

# Check StatefulSet
kubectl get sts
```

```
kubectl describe sts web
```

Job

bash

```
# Create Job
kubectl create job pi --image=perl -- perl -Mbignum=bpi -wle
'print bpi(2000)'

# Check Job
kubectl get jobs
kubectl describe job pi
kubectl logs job/pi
```

backoffLimit

> *"Retry failed pods 4 times"*

activeDeadlineSeconds

> *"Give up if it takes more than 10 minutes"*

ttlSecondsAfterFinished

> *"Delete the job object after it's done in 30 minutes"*

successfulJobsHistoryLimit / failedJobsHistoryLimit (for CronJobs)

> *"Only keep last 3 successful jobs"*

startingDeadlineSeconds (CronJob)

> *"If a scheduled time was missed, give it 30 seconds to catch up"*

CronJob

bash

```
# Create CronJob
kubectl create cronjob hello --image=busybox --schedule="*/1 * * *
*" -- echo "Hello World"
```

```
# Check CronJob
kubectl get cronjobs

kubectl describe cronjob hello
```

Cron Schedule Syntax

```
* * * * *
| | | | |
| | | | └──── day of week (0-6) (Sunday=0)
| | | └────── month (1-12)
| | └──────── day of month (1-31)
| └────────── hour (0-23)
└──────────── minute (0-59)
```

Common Schedules:

1. `0 * * * *` - Every hour at minute 0
2. 0-4 * * * * - Every minute from 0 through 4 of every hour (runs everyminute for 5 minutes, every hour)
3. `*/15 * * * *` - Every 15 minutes but only at say 10: 15, 10:30, etc. if You want to run it every 15th minute of every hour then do “15 * * * *” instead.
4. `0 0 * * *` - Daily at midnight
5. `0 0 * * 0` - Weekly on Sunday midnight
6. `0 0 1 * *` - Monthly on the 1st at midnight

Multi-Container Pod Patterns

Sidecar Pattern

- Main container handles primary functionality
- Sidecar container enhances/extends main container
- Example: Log collector, file sync, proxy

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-sidecar
spec:
  containers:
  - name: app
    image: app:1.0
```

```
  - name: log-collector
    image: log-collector:1.0
    volumeMounts:
    - name: logs
      mountPath: /var/log
  volumes:
  - name: logs
    emptyDir: {}
```

Init Container Pattern

- Runs before app containers
- Must complete successfully before app containers start
- Used for setup tasks, dependency checks, delays

yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  initContainers:
  - name: setup
    image: busybox
    command: ['sh', '-c', 'until ping -c 1 database; do echo
waiting; sleep 2; done;']
  containers:
  - name: app
    image: app:1.0
```

Ambassador Pattern

- Proxy for accessing external services
- Simplifies application network access

Adapter Pattern

- Standardizes application output
- Transforms data to common format

Volumes

Volume Types

- **emptyDir**: Temporary storage for pod lifetime
- **hostPath**: Mounts file/directory from host node

- **configMap**: Injects config data
- **secret**: Stores sensitive data
- **persistentVolumeClaim**: Uses pre-provisioned or dynamically provisioned storage
- **downwardAPI**: Exposes pod/container info
- **projected**: Maps multiple volume sources
- **nfs**: Network File System share
- **csi**: Container Storage Interface

PersistentVolume (PV)

yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-example
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain  # or Delete or Recycle
  storageClassName: standard
  hostPath:
    path: /data
```

Access Modes:

- **ReadWriteOnce (RWO)**: Single node read/write
- **ReadOnlyMany (ROX)**: Multiple nodes read-only
- **ReadWriteMany (RWX)**: Multiple nodes read/write

Reclaim Policies:

- **Retain**: Manual reclamation
- **Delete**: Delete PV and storage
- **Recycle**: Basic scrub (deprecated)

PersistentVolumeClaim (PVC)

yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-example
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```yaml
      storage: 5Gi
  storageClassName: standard
```

Using PVC in Pod

yaml
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod
spec:
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-example
```

Storage Classes

yaml
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Delete
allowVolumeExpansion: true
```

Common CKAD Commands

bash
```bash
# Context and namespace
kubectl config use-context <context>
kubectl config set-context --current --namespace=<namespace>

# Create resources
kubectl create -f manifest.yaml
kubectl apply -f manifest.yaml

# View resources
```

```
kubectl get pods,svc,deploy,cm,secrets
kubectl describe pod <pod>
kubectl logs [-f] <pod> [-c <container>]

# Debug
kubectl exec -it <pod> [-c <container>] -- /bin/sh
kubectl port-forward <pod> 8080:80

# Delete resources
kubectl delete pod <pod> [--force] [--grace-period=0]

# Editing resources
kubectl edit deploy <deployment>
kubectl set resources deployment <deploy> -c=<container>
--limits=cpu=200m,memory=512Mi
```

Exam Tips

- Use aliases: `alias k=kubectl`
- Use `kubectl explain` for quick reference
- Save time with `kubectl run/create` generators
- Use imperative commands with `--dry-run=client -o yaml`
- Tab completion is your friend
- Remember CRUD operations for all resource types

Kubernetes CKAD Cheatsheet: Part 2

Deployment Strategies

Blue/Green Deployment

- Two identical environments: Blue (current) and Green (new)
- Switch traffic at once using Service selector

yaml

```yaml
# Green deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-green
  labels:
    app: myapp
    version: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      version: green
  template:
    metadata:
      labels:
        app: myapp
        version: green
    spec:
      containers:
      - name: app
        image: myapp:2.0
```

yaml

```yaml
# Service (initially pointing to blue)
apiVersion: v1
kind: Service
metadata:
  name: myapp-svc
spec:
  selector:
    app: myapp
    version: blue  # Change to 'green' to switch
  ports:
  - port: 80
    targetPort: 8080
```

Commands:

bash

```bash
# Create both deployments
```

```
kubectl apply -f blue-deployment.yaml
kubectl apply -f green-deployment.yaml

# Switch traffic to green
kubectl patch service myapp-svc -p
'{"spec":{"selector":{"version":"green"}}}'

# After verification, delete blue
kubectl delete deployment app-blue
```

Canary Deployment

- Gradual traffic shifting using multiple deployments with same labels
- Control traffic percentage with replica counts

yaml

```yaml
# Stable deployment (90% traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-stable
spec:
  replicas: 9  # 9 pods = 90% traffic
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: app
        image: myapp:1.0
```

yaml

```yaml
# Canary deployment (10% traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-canary
spec:
  replicas: 1  # 1 pod = 10% traffic
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
```

```yaml
      labels:
        app: myapp
    spec:
      containers:
      - name: app
        image: myapp:2.0
```

Commands:

bash
```bash
# Create both deployments
kubectl apply -f stable.yaml
kubectl apply -f canary.yaml

# Increase canary traffic by scaling
kubectl scale deployment app-canary --replicas=3  # Now 25% traffic

# Full rollout - scale up canary, scale down stable
kubectl scale deployment app-canary --replicas=10
kubectl scale deployment app-stable --replicas=0
```

Rolling Updates with Deployments

Deployment Update Strategies

yaml
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 5
  strategy:
    type: RollingUpdate  # or 'Recreate'
    rollingUpdate:
      maxSurge: 1        # Max pods above desired count
      maxUnavailable: 1  # Max pods unavailable during update
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: app
```

```
        image: myapp:1.0
```

Rolling Update Commands

bash

```
# Update image
kubectl set image deployment/myapp app=myapp:2.0

# Check rollout status
kubectl rollout status deployment/myapp

# View rollout history
kubectl rollout history deployment/myapp

# Undo rollout
kubectl rollout undo deployment/myapp [--to-revision=2]

# Pause/resume rollout
kubectl rollout pause deployment/myapp
kubectl rollout resume deployment/myapp

# Record command for history (deprecated)
kubectl set image deployment/myapp app=myapp:2.0 --record
```

Helm Package Manager

Basic Commands

bash

```
# Add a repository
helm repo add bitnami https://charts.bitnami.com/bitnami

# Update repos
helm repo update

# Search for charts
helm search repo nginx

# Install a chart
helm install myrelease bitnami/nginx

# Install with custom values
helm install myrelease bitnami/nginx -f values.yaml

# Install with specific version
helm install myrelease bitnami/nginx --version 9.3.0

# Override values inline
```

```
helm install myrelease bitnami/nginx --set
replicaCount=3,service.type=LoadBalancer

# List releases
helm list

# Get release status
helm status myrelease

# See deployed manifests
helm get manifest myrelease

# Upgrade a release
helm upgrade myrelease bitnami/nginx --reuse-values --set
replicaCount=5

# Rollback a release
helm rollback myrelease 1

# Uninstall a release
helm uninstall myrelease
```

Creating Custom Charts

bash

```
# Create a new chart
helm create mychart

# Package a chart
helm package mychart

# Lint a chart
helm lint mychart

# Install from local directory
helm install myrelease ./mychart

# Install with debug
helm install myrelease ./mychart --debug --dry-run
```

Kustomize

Directory Structure

```
base/
  kustomization.yaml
  deployment.yaml
```

```
  service.yaml
overlays/
  dev/
    kustomization.yaml
    config.yaml  # Dev-specific overrides
  prod/
    kustomization.yaml
    config.yaml  # Prod-specific overrides
```

Base kustomization.yaml

yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
  - service.yaml
commonLabels:
  app: myapp
```

Overlay kustomization.yaml (dev)

yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
bases:
  - ../../base
namePrefix: dev-
commonLabels:
  env: dev
patchesStrategicMerge:
  - config.yaml
images:
  - name: myapp
    newName: myapp
    newTag: dev-1.0
replicas:
  - name: myapp
    count: 1
```

Commands

bash

```bash
# View generated resources from base
kubectl kustomize base/

# Apply kustomized resources
kubectl apply -k base/
```

```
kubectl apply -k overlays/dev/
kubectl apply -k overlays/prod/

# Generate resources and save to file
kubectl kustomize overlays/dev/ > dev-resources.yaml
```

Probes and Health Checks

Types of Probes

- **Liveness**: Detects if container is running properly
- **Readiness**: Detects if container can receive traffic
- **Startup**: Detects if application has started (similar to liveness but only during startup)

Probe Methods

- **HTTP GET**: Sends HTTP request to specified path
- **TCP Socket**: Tries to establish TCP connection
- **Exec**: Executes command in container

yaml
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-demo
spec:
  containers:
  - name: app
    image: myapp:1.0
    ports:
    - containerPort: 8080
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 10
      timeoutSeconds: 5
      failureThreshold: 3
    readinessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5
    startupProbe:
      httpGet:
```

```
    path: /startup
    port: 8080
  failureThreshold: 30
  periodSeconds: 10
```

Monitoring Applications

Resource Monitoring

bash

```
# View node resource usage
kubectl top node

# View pod resource usage
kubectl top pod [-n namespace]

# Resource usage with labels
kubectl top pod -l app=myapp
```

Events

bash

```
# View events sorted by timestamp
kubectl get events --sort-by='.lastTimestamp'

# View events for specific resource
kubectl describe pod <podname>
```

API Server Requests

bash

```
# View API resources
kubectl api-resources

# Explain resource fields
kubectl explain pod.spec.containers

# API server call with verbose output
kubectl get pods -v=6

# Check API connection
kubectl auth can-i create pods
```

Container Logs

Basic Logging

bash

```
# Get logs from pod
kubectl logs mypod

# Get logs from specific container in multi-container pod
kubectl logs mypod -c mycontainer

# Follow logs (stream)
kubectl logs -f mypod

# Get recent logs
kubectl logs --tail=100 mypod

# Get logs with timestamps
kubectl logs --timestamps=true mypod

# Get logs since duration
kubectl logs --since=1h mypod

# Get logs from all pods with label
kubectl logs -l app=myapp --all-containers=true

# Output logs to file
kubectl logs mypod > pod.log
```

Log Aggregation

Common pattern: Sidecar container collects logs from main container

yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: logging-pod
spec:
  containers:
  - name: app
    image: myapp:1.0
    volumeMounts:
    - name: log-storage
      mountPath: /var/log/app
  - name: log-collector
    image: log-collector:1.0
    volumeMounts:
    - name: log-storage
      mountPath: /var/log/app
```

```
  volumes:
  - name: log-storage
    emptyDir: {}
```

Debugging in Kubernetes

Pod Troubleshooting

bash

```
# Check pod status
kubectl get pod mypod -o wide

# Describe pod for events and config
kubectl describe pod mypod

# Check logs
kubectl logs mypod [-c container]

# Execute commands in pod
kubectl exec -it mypod -- sh

# Check resource usage
kubectl top pod mypod

# Port forwarding for direct testing
kubectl port-forward mypod 8080:80
```

Common Debugging Steps

1. **Pod stuck in Pending**
   - Check cluster resources: `kubectl describe node`
   - Check PVC binding: `kubectl get pvc`
   - Check node taints/affinity: `kubectl describe node`
2. **Pod stuck in ContainerCreating**
   - Volume mount issues: `kubectl describe pod`
   - Image pull issues: `kubectl describe pod`
   - Check kubelet logs on node
3. **Pod stuck in CrashLoopBackOff**
   - Check container logs: `kubectl logs mypod`
   - Check previous terminated container: `kubectl logs mypod --previous`
   - Check liveness/readiness probes
   - Exec into pod to debug: `kubectl exec -it mypod -- sh`
4. **Pod running but not working**
   - Check service endpoints: `kubectl get endpoints myservice`
   - Verify network policies: `kubectl get netpol`

- Test connectivity: `kubectl exec -it testpod -- curl http://service`

Debug Tools

bash

```
# Create debug container attached to pod's namespace
kubectl debug mypod -it --image=busybox --share-processes
--copy-to=debug-mypod

# Create debug pod on node
kubectl debug node/mynode -it --image=ubuntu

# Copy files between pod and local machine
kubectl cp mypod:/path/to/file ./local-file
kubectl cp ./local-file mypod:/path/to/file

# Temporary debug pod using ephemeral containers (kubectl v1.23+)
kubectl debug -it mypod --image=busybox --target=mypod
```

Restart Resources

bash

```
# Restart deployment without changing definition
kubectl rollout restart deployment/myapp

# Delete pod (will be recreated by controller)
kubectl delete pod mypod

# Scale down/up for restart
kubectl scale deployment myapp --replicas=0
kubectl scale deployment myapp --replicas=3
```

Quick Testing Commands

bash

```
# Test DNS resolution
kubectl run -it --rm debug --image=busybox -- nslookup
kubernetes.default

# Test network connectivity
kubectl run -it --rm debug --image=busybox -- wget -O-
http://myservice:8080

# Create temporary debugging pod
kubectl run debug --rm -it --image=ubuntu -- bash
```

Custom Resources & Operators

Custom Resource Definitions (CRDs)

yaml

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: backups.mycompany.com
spec:
  group: mycompany.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                backupType:
                  type: string
                schedule:
                  type: string
  scope: Namespaced
  names:
    plural: backups
    singular: backup
    kind: Backup
    shortNames:
    - bk
```

Working with Custom Resources

bash

```bash
# List CRDs
kubectl get crd

# Get specific CRD details
kubectl describe crd backups.mycompany.com

# Create custom resource
kubectl apply -f backup.yaml

# List custom resources
kubectl get backups
```

Operators

- Operators = Controller + CRD
- Common operators: Prometheus, PostgreSQL, MongoDB, etc.

bash
```
# Install Operator Lifecycle Manager (OLM)
curl -sL
https://github.com/operator-framework/operator-lifecycle-manager/releases/download/v0.20.0/install.sh | bash -s v0.20.0

# Install operator via OLM (example)
kubectl create -f https://operatorhub.io/install/prometheus.yaml

# Check operator status
kubectl get csv -n operators
```

Authentication, Authorization, Admission Control

Authentication Methods

- X.509 Client Certificates
- Static Token File
- Bootstrap Tokens
- Service Account Tokens
- OpenID Connect (OIDC)
- Webhook Token Authentication

ServiceAccounts

yaml
```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-sa
  namespace: default
```

bash
```
# Create service account
kubectl create serviceaccount app-sa

# Get service account
kubectl get sa app-sa

# Use service account in pod
kubectl run nginx --image=nginx --serviceaccount=app-sa
```

RBAC: Role and RoleBinding

yaml

```yaml
# Role defines permissions
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list", "watch"]
```

yaml

```yaml
# RoleBinding assigns role to users/groups/serviceaccounts
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: app-sa
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

ClusterRole and ClusterRoleBinding

- Similar to Role/RoleBinding but cluster-wide (not namespaced)

yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
```

```yaml
  name: read-secrets-global
subjects:
- kind: ServiceAccount
  name: app-sa
  namespace: default
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC Testing

bash

```
# Check permissions
kubectl auth can-i get pods
--as=system:serviceaccount:default:app-sa

# Check permissions in namespace
kubectl auth can-i list deployments --namespace dev
--as=system:serviceaccount:default:app-sa
```

Resource Management

Resource Requests and Limits

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource-demo
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"      # 0.25 CPU cores
      limits:
        memory: "128Mi"
        cpu: "500m"      # 0.5 CPU cores
```

ResourceQuota

yaml

```yaml
apiVersion: v1
kind: ResourceQuota
```

```yaml
metadata:
  name: compute-quota
  namespace: dev
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 4Gi
    limits.cpu: "8"
    limits.memory: 8Gi
    count/deployments.apps: "5"
    count/replicasets.apps: "10"
```

bash

```bash
# Create quota
kubectl create quota compute-quota
--hard=pods=10,requests.cpu=4,requests.memory=4Gi

# Check quota usage
kubectl describe quota compute-quota -n dev
```

LimitRange

yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: default-limits
  namespace: dev
spec:
  limits:
  - default:            # Default limits
      cpu: 500m
      memory: 256Mi
    defaultRequest:     # Default requests
      cpu: 100m
      memory: 128Mi
    type: Container
```

bash

```bash
# Create limit range
kubectl create -f limitrange.yaml

# Check limit range
kubectl describe limitrange default-limits -n dev
```

ConfigMaps

Creating ConfigMaps

bash

```bash
# From literal values
kubectl create configmap app-config --from-literal=DB_URL=mysql
--from-literal=DB_PORT=3306

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

# From directory
kubectl create configmap app-config --from-file=config-dir/

# From env file
kubectl create configmap app-config --from-env-file=config.env
```

yaml

```yaml
# Declarative creation
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  DB_URL: mysql
  DB_PORT: "3306"
  config.json: |
    {
      "environment": "dev",
      "debug": true
    }
```

Using ConfigMaps

yaml

```yaml
# Environment variables
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
  - name: app
    image: myapp
    env:
    - name: DB_URL
      valueFrom:
```

```
        configMapKeyRef:
          name: app-config
          key: DB_URL
    # Load all keys as env vars
    envFrom:
    - configMapRef:
        name: app-config
```

yaml

```
# Volume mount
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
  - name: app
    image: myapp
    volumeMounts:
    - name: config-vol
      mountPath: /etc/config
  volumes:
  - name: config-vol
    configMap:
      name: app-config
      # Optional: specific items only
      items:
      - key: config.json
        path: app/config.json
```

Secrets

Types of Secrets

- **Opaque**: Generic key-value pairs
- **kubernetes.io/tls**: TLS certificates
- **kubernetes.io/dockerconfigjson**: Docker registry credentials
- **kubernetes.io/service-account-token**: Service account token

Creating Secrets

bash

```
# From literal values
kubectl create secret generic db-creds
--from-literal=username=admin --from-literal=password=secret

# From files
```

```
kubectl create secret generic tls-certs --from-file=cert.pem
--from-file=key.pem

# Docker registry secret
kubectl create secret docker-registry regcred
--docker-server=https://index.docker.io/v1/ --docker-username=user
--docker-password=pass --docker-email=user@example.com

# TLS secret
kubectl create secret tls tls-secret --cert=cert.pem --key=key.pem
```

yaml

```
# Declarative creation (values must be base64 encoded)
apiVersion: v1
kind: Secret
metadata:
  name: db-creds
type: Opaque
data:
  username: YWRtaW4=  # echo -n "admin" | base64
  password: c2VjcmV0  # echo -n "secret" | base64
```

Using Secrets

yaml

```
# Environment variables
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
  - name: app
    image: myapp
    env:
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-creds
          key: username
    # Load all keys as env vars
    envFrom:
    - secretRef:
        name: db-creds
```

yaml

```
# Volume mount
apiVersion: v1
```

```yaml
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
  - name: app
    image: myapp
    volumeMounts:
    - name: secret-vol
      mountPath: /etc/secrets
      readOnly: true
  volumes:
  - name: secret-vol
    secret:
      secretName: db-creds
```

Security Contexts and Pod Security

Security Context

yaml

```yaml
# Pod-level security context
apiVersion: v1
kind: Pod
metadata:
  name: security-pod
spec:
  securityContext:
    fsGroup: 2000           # GID for mounted volumes
    runAsNonRoot: true      # Don't run as root
    runAsUser: 1000         # UID to run as
    runAsGroup: 3000        # GID to run as
  containers:
  - name: app
    image: myapp
    # Container-level security context (overrides pod-level)
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        add: ["NET_ADMIN"]
        drop: ["ALL"]
      privileged: false
      readOnlyRootFilesystem: true
      runAsUser: 1001       # Override pod-level UID
```

Pod Security Policies (Deprecated)

Now replaced by Pod Security Admission Controller with three profiles:

- **Privileged**: Unrestricted policy (default)
- **Baseline**: Prevents known privilege escalations
- **Restricted**: Heavily restricted policy

bash

```
# Set namespace to enforce restricted profile
kubectl label namespace default
pod-security.kubernetes.io/enforce=restricted
```

Network Policies

Basic Network Policy

yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
  namespace: default
spec:
  podSelector:                    # Applies to pods with label
role=db
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:                # Allow from pods with label
role=frontend
        matchLabels:
          role: frontend
    - namespaceSelector:          # Allow from monitoring namespace
        matchLabels:
          name: monitoring
    ports:
    - protocol: TCP
      port: 3306
  egress:
  - to:
    - ipBlock:                    # Allow to specific CIDR
        cidr: 10.0.0.0/24
        except:
        - 10.0.0.5/32
```

```
    ports:
    - protocol: TCP
      port: 53                    # DNS
```

Commands

bash

```
# Create network policy
kubectl apply -f network-policy.yaml

# Get network policies
kubectl get networkpolicy

# Describe network policy
kubectl describe networkpolicy db-policy
```

Testing Network Policies

bash

```
# Create test pod
kubectl run test-pod --image=busybox --rm -it -- sh

# Test connection
wget -q -O- http://service:port
nc -zv service port
```

Services

Service Types

- **ClusterIP**: Internal-only IP (default)
- **NodePort**: Exposes port on each node's IP
- **LoadBalancer**: External load balancer (requires cloud provider)
- **ExternalName**: DNS CNAME record

Creating Services

bash

```
# Expose deployment
kubectl expose deployment nginx --port=80 --target-port=8080
--type=ClusterIP

# Create NodePort service
kubectl expose deployment nginx --port=80 --target-port=8080
--type=NodePort

# Create LoadBalancer service
```

```
kubectl expose deployment nginx --port=80 --target-port=8080
--type=LoadBalancer
```

yaml

```yaml
# ClusterIP service
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
  - protocol: TCP
    port: 80          # Service port
    targetPort: 8080  # Container port
```

yaml

```yaml
# NodePort service
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30007   # Optional: 30000-32767
```

yaml

```yaml
# LoadBalancer service
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
```

yaml

```yaml
# ExternalName service
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-db
spec:
  type: ExternalName
  externalName: db.example.com
```

Headless Service

yaml
```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless-service
spec:
  clusterIP: None     # Headless
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
```

Service Discovery

bash
```bash
# Through environment variables
env | grep SERVICE

# Through DNS
# <service-name>.<namespace>.svc.cluster.local
nslookup my-service.default.svc.cluster.local
```

Troubleshooting Services

bash
```bash
# Check service
kubectl get svc my-service

# Check endpoints (should match pod IPs)
kubectl get endpoints my-service

# Check pod labels
kubectl get pods --show-labels

# DNS resolution in cluster
kubectl run -it --rm dns-test --image=busybox -- nslookup
my-service
```

```bash
# Test direct connection to pods
kubectl get pod -o wide

# Then connect to pod IP:port
```

Ingress

Ingress Controller

Common controllers: Nginx, Traefik, HAProxy, etc.

bash

```bash
# Install Nginx Ingress Controller
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.0/deploy/static/provider/cloud/deploy.yaml
```

Basic Ingress

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: basic-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: myapp-service
            port:
              number: 80
```

Path-Based Routing

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```yaml
  name: path-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: example.com
    http:
      paths:
      - path: /app1
        pathType: Prefix
        backend:
          service:
            name: app1-service
            port:
              number: 80
      - path: /app2
        pathType: Prefix
        backend:
          service:
            name: app2-service
            port:
              number: 80
```

Name-Based Virtual Hosting

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: app1.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app1-service
            port:
              number: 80
  - host: app2.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
```

```yaml
          service:
            name: app2-service
            port:
              number: 80
```

TLS Termination

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - secure.example.com
    secretName: tls-secret  # kubectl create secret tls tls-secret --cert=tls.crt --key=tls.key
  rules:
  - host: secure.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: secure-service
            port:
              number: 80
```

Common Ingress Annotations

yaml

```yaml
metadata:
  annotations:
    # Nginx specific
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/proxy-body-size: "10m"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "30"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "60"

    # Common
    kubernetes.io/ingress.class: "nginx"  # deprecated, use spec.ingressClassName
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
```

Troubleshooting Ingress

bash

```bash
# Check Ingress
kubectl get ingress

# Check Ingress controller pods
kubectl get pods -n ingress-nginx

# Check Ingress controller logs
kubectl logs -n ingress-nginx deploy/ingress-nginx-controller

# Check if services and endpoints exist
kubectl get svc,ep

# Test using curl
curl -H "Host: myapp.example.com" http://<ingress-ip>
```