Table of Contents

[**1) Application Design and Build** 2](#_Toc193397045)

[**1.1 Choose and use the right workload resource** 2](#_Toc193397046)

[**1.2 Multi-Container Pod Design Patterns** 4](#_Toc193397047)

[**1.3 Persistent and Ephemeral Volumes** 6](#_Toc193397048)

[**2) Application Deployments** 7](#_Toc193397049)

[**2.1 Deployments and Rolling Updates** 7](#_Toc193397050)

[**2.2 Helm Package Manager** 9](#_Toc193397051)

[**3) Application Environment, Configuration and Security** 10](#_Toc193397052)

[**3.1 Requests, Limits, Quotas** 10](#_Toc193397053)

[**3.2 ConfigMaps** 11](#_Toc193397054)

[**3.3 Define Resource Requirements** 13](#_Toc193397055)

[**3.4 Create & Consume Secrets** 13](#_Toc193397056)

[**3.5 ServiceAccounts** 14](#_Toc193397057)

[**3.6 Application Security (SecurityContexts, Capabilities)** 16](#_Toc193397058)

**1) Application Design and Build**

**1.1 Choose and use the right workload resource**

**Deployments**

* **Purpose**: For stateless applications that need scaling and rolling updates
* **Use when**: Running web servers, API services, or stateless microservices
* **Key commands**:

# Create a deployment

kubectl create deployment nginx --image=nginx:1.19

# View deployments

kubectl get deployments

# Scale a deployment

kubectl scale deployment nginx --replicas=3

* **Example YAML**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.19

ports:

- containerPort: 80

**StatefulSets**

* **Purpose**: For stateful applications requiring stable network identifiers and persistent storage
* **Use when**: Running databases, distributed systems like Kafka, ElasticSearch
* **Key commands**:

# Create a statefulset (usually from YAML)

kubectl apply -f statefulset.yaml

# View statefulsets

kubectl get statefulsets

**DaemonSets**

* **Purpose**: Ensures that a copy of the pod runs on each node
* **Use when**: You need monitoring agents, log collectors, or node-level services
* **Key commands**:

# View daemonsets

kubectl get daemonsets

# Create a daemonset (usually from YAML)

kubectl apply -f daemonset.yaml

* **Example YAML**:

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd

spec:

selector:

matchLabels:

app: fluentd

template:

metadata:

labels:

app: fluentd

spec:

containers:

- name: fluentd

image: fluentd:v1.7

**Jobs**

* **Purpose**: For one-off tasks that run to completion
* **Use when**: Batch processing, data migrations, backups
* **Key commands**:

# Create a job

kubectl create job hello --image=busybox -- echo "Hello World"

# View jobs

kubectl get jobs

**CronJobs**

* **Purpose**: Jobs that run on a time-based schedule
* **Use when**: Scheduled backups, report generation, periodic clean-up tasks
* **Key commands**:

# Create a cronjob that runs every minute

kubectl create cronjob hello --image=busybox --schedule="\*/1 \* \* \* \*" -- echo "Hello World"

# View cronjobs

kubectl get cronjobs

* **Example YAML**:

apiVersion: batch/v1

kind: CronJob

metadata:

name: backup-job

spec:

schedule: "0 1 \* \* \*" # Every day at 1 AM

jobTemplate:

spec:

template:

spec:

containers:

- name: backup

image: backup-tool:latest

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

restartPolicy: OnFailure

**1.2 Multi-Container Pod Design Patterns**

**Sidecar Pattern**

* **Purpose**: Enhances the main container with additional functionality
* **Example use cases**: Log shipping, file synchronization, watchers
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: web-with-sidecar

spec:

containers:

- name: web

image: nginx:1.19

ports:

- containerPort: 80

- name: log-collector

image: fluentd:v1.7

volumeMounts:

- name: logs

mountPath: /var/log/nginx

volumes:

- name: logs

emptyDir: {}

**Ambassador Pattern**

* **Purpose**: Proxies network connections to the outside world
* **Example use cases**: Database proxies, connection pooling, service discovery
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: app-with-ambassador

spec:

containers:

- name: app

image: myapp:1.0

env:

- name: REDIS\_HOST

value: localhost

- name: redis-ambassador

image: redis-proxy:latest

**Adapter Pattern**

* **Purpose**: Standardizes or transforms the output of the main container
* **Example use cases**: Monitoring adapters, log formatters
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: app-with-adapter

spec:

containers:

- name: app

image: myapp:1.0

- name: log-adapter

image: log-format-adapter:latest

volumeMounts:

- name: logs

mountPath: /app/logs

volumes:

- name: logs

emptyDir: {}

**Init Container Pattern**

* **Purpose**: Runs before the main container and completes setup
* **Example use cases**: Schema migration, configuration setup, dependency checks
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: myapp-pod

spec:

initContainers:

- name: init-db-check

image: busybox:1.28

command: ['sh', '-c', 'until ping -c 1 mysql; do echo waiting for mysql; sleep 2; done;']

containers:

- name: myapp

image: myapp:1.0

**1.3 Persistent and Ephemeral Volumes**

**Ephemeral Volumes (emptyDir)**

* **Purpose**: Temporary storage that exists for the lifetime of the pod
* **Use when**: Sharing data between containers, temporary processing space
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: shared-data-pod

spec:

containers:

- name: container-1

image: busybox

volumeMounts:

- name: shared-data

mountPath: /data1

- name: container-2

image: busybox

volumeMounts:

- name: shared-data

mountPath: /data2

volumes:

- name: shared-data

emptyDir: {}

**Persistent Volumes (PV) and Claims (PVC)**

* **Purpose**: Storage that exists independent of pods
* **Use when**: Databases, file storage, shared application data
* **Key commands**:

# Create a persistent volume claim

kubectl apply -f pvc.yaml

# View persistent volumes and claims

kubectl get pv

kubectl get pvc

* **Example YAML - PVC**:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-data

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5Gi

* **Pod using PVC**:

apiVersion: v1

kind: Pod

metadata:

name: mysql-pod

spec:

containers:

- name: mysql

image: mysql:5.7

env:

- name: MYSQL\_ROOT\_PASSWORD

value: password

volumeMounts:

- name: mysql-storage

mountPath: /var/lib/mysql

volumes:

- name: mysql-storage

persistentVolumeClaim:

claimName: mysql-data

**StorageClasses**

* **Purpose**: Define different classes of storage with specific provisioners
* **Use when**: You need different storage performance tiers or access modes
* **Example YAML**:

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: fast

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp2

**2) Application Deployments**

**2.1 Deployments and Rolling Updates**

**Rolling Updates**

* **Purpose**: Update applications with zero downtime
* **Key commands**:

# Update a deployment image

kubectl set image deployment/nginx-deployment nginx=nginx:1.20

# Check rollout status

kubectl rollout status deployment/nginx-deployment

# Rollback an update

kubectl rollout undo deployment/nginx-deployment

# View rollout history

kubectl rollout history deployment/nginx-deployment

* **Update Strategy YAML**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 1 # Max pods above desired count during update

maxUnavailable: 0 # Min pods available during update

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.19

**Canary Deployments**

* **Purpose**: Test a new version with a subset of users before full rollout
* **Implementation**: Use two deployments with the same selector but different labels
* **Example YAML - New Version (Canary)**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-canary

spec:

replicas: 1 # Low replica count for canary

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

version: canary

spec:

containers:

- name: nginx

image: nginx:1.20

**2.2 Helm Package Manager**

**Helm Basics**

* **Purpose**: Package manager for Kubernetes applications
* **Key concepts**: Charts (packages), Releases (installed instances), Repositories
* **Key commands**:

# Add a repository

helm repo add bitnami https://charts.bitnami.com/bitnami

# Update repositories

helm repo update

# Search for charts

helm search repo nginx

# Install a chart

helm install my-nginx bitnami/nginx

# List installed releases

helm list

# Upgrade a release

helm upgrade my-nginx bitnami/nginx --set replicaCount=3

# Rollback a release

helm rollback my-nginx 1

# Uninstall a release

helm uninstall my-nginx

**Creating Custom Helm Charts**

* **Key commands**:

# Create a new chart

helm create mychart

# Package a chart

helm package mychart

# Install from local chart

helm install my-release ./mychart

* **Chart.yaml example**:

apiVersion: v2

name: mychart

description: A Helm chart for my application

type: application

version: 0.1.0

appVersion: 1.0.0

* **values.yaml example**:

replicaCount: 1

image:

repository: nginx

tag: 1.19

pullPolicy: IfNotPresent

service:

type: ClusterIP

port: 80

**3) Application Environment, Configuration and Security**

**3.1 Requests, Limits, Quotas**

**Resource Requests and Limits**

* **Purpose**: Define minimum (requests) and maximum (limits) resources for containers
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: resource-demo

spec:

containers:

- name: app

image: nginx

resources:

requests:

memory: "128Mi"

cpu: "100m"

limits:

memory: "256Mi"

cpu: "500m"

**Resource Quotas**

* **Purpose**: Limit resource consumption per namespace
* **Example YAML**:

apiVersion: v1

kind: ResourceQuota

metadata:

name: compute-quota

namespace: dev

spec:

hard:

pods: "10"

requests.cpu: "4"

requests.memory: 8Gi

limits.cpu: "8"

limits.memory: 16Gi

* **Key commands**:

# Create a quota

kubectl apply -f quota.yaml

# View quotas

kubectl get resourcequota

**LimitRanges**

* **Purpose**: Set default resource limits for pods in a namespace
* **Example YAML**:

apiVersion: v1

kind: LimitRange

metadata:

name: default-limits

namespace: dev

spec:

limits:

- default:

cpu: "500m"

memory: "256Mi"

defaultRequest:

cpu: "100m"

memory: "128Mi"

type: Container

**3.2 ConfigMaps**

**ConfigMaps**

* **Purpose**: Store non-sensitive configuration data
* **Key commands**:

# Create configmap from literal values

kubectl create configmap app-config --from-literal=DB\_HOST=mysql --from-literal=DB\_PORT=3306

# Create configmap from file

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

# View configmaps

kubectl get configmaps

kubectl describe configmap app-config

* **Example YAML**:

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

DB\_HOST: "mysql"

DB\_PORT: "3306"

config.properties: |

app.name=MyApp

app.log.level=INFO

**Using ConfigMaps in Pods**

* **Environment Variables**:

apiVersion: v1

kind: Pod

metadata:

name: app-pod

spec:

containers:

- name: app

image: myapp:1.0

env:

- name: DB\_HOST

valueFrom:

configMapKeyRef:

name: app-config

key: DB\_HOST

- name: DB\_PORT

valueFrom:

configMapKeyRef:

name: app-config

key: DB\_PORT

* **Volume Mounts**:

apiVersion: v1

kind: Pod

metadata:

name: app-pod

spec:

containers:

- name: app

image: myapp:1.0

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: app-config

**3.3 Define Resource Requirements**

**Best Practices for Resource Allocation**

* Start with metrics from local testing to establish baseline requirements
* Monitor actual usage in staging environment
* Set requests slightly above average usage
* Set limits based on acceptable peak usage
* Leave headroom for spikes and garbage collection

**Quality of Service (QoS) Classes**

* **Guaranteed**: When requests equal limits (highest priority)
* **Burstable**: When requests are set but limits are higher or not set
* **BestEffort**: When neither requests nor limits are set (lowest priority)

**3.4 Create & Consume Secrets**

**Secrets**

* **Purpose**: Store sensitive information like passwords and tokens
* **Key commands**:

# Create secret from literal values

kubectl create secret generic db-creds --from-literal=username=admin --from-literal=password=secret

# Create secret from file

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

# View secrets

kubectl get secrets

kubectl describe secret db-creds

* **Example YAML**:

apiVersion: v1

kind: Secret

metadata:

name: db-creds

type: Opaque

data:

username: YWRtaW4= # base64 encoded "admin"

password: c2VjcmV0 # base64 encoded "secret"

**Using Secrets in Pods**

* **Environment Variables**:

apiVersion: v1

kind: Pod

metadata:

name: db-client

spec:

containers:

- name: client

image: db-client:1.0

env:

- name: DB\_USERNAME

valueFrom:

secretKeyRef:

name: db-creds

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-creds

key: password

* **Volume Mounts**:

apiVersion: v1

kind: Pod

metadata:

name: db-client

spec:

containers:

- name: client

image: db-client:1.0

volumeMounts:

- name: creds-volume

mountPath: /etc/creds

readOnly: true

volumes:

- name: creds-volume

secret:

secretName: db-creds

**3.5 ServiceAccounts**

**ServiceAccounts**

* **Purpose**: Identity for processes running in a pod
* **Key commands**:

# Create service account

kubectl create serviceaccount app-sa

# View service accounts

kubectl get serviceaccounts

kubectl describe serviceaccount app-sa

* **Example YAML**:

apiVersion: v1

kind: ServiceAccount

metadata:

name: app-sa

* **Pod Using ServiceAccount**:

apiVersion: v1

kind: Pod

metadata:

name: app-pod

spec:

serviceAccountName: app-sa

containers:

- name: app

image: myapp:1.0

**RBAC for ServiceAccounts**

* **Role Example**:

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: default

name: pod-reader

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list", "watch"]

* **RoleBinding Example**:

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

**3.6 Application Security (SecurityContexts, Capabilities)**

**Pod SecurityContext**

* **Purpose**: Define privilege and access control settings for pods and containers
* **Example YAML**:

apiVersion: v1

kind: Pod

metadata:

name: security-context-pod

spec:

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: 2000

containers:

- name: app

image: security-app:1.0

securityContext:

allowPrivilegeEscalation: false

capabilities:

add: ["NET\_ADMIN"]

drop: ["ALL"]

**Key Security Settings**

* runAsUser/runAsGroup: Set UID/GID for running processes
* fsGroup: Set group ID for volume access
* allowPrivilegeEscalation: Control if process can gain more privileges
* readOnlyRootFilesystem: Mount root filesystem as read-only
* capabilities: Linux capabilities to add or drop

**Pod Security Policies (Deprecated but Still Important to Understand)**

* Control security-sensitive aspects of pod specifications
* Define what a pod can or cannot do
* Enforce security best practices cluster-wide

**Security Best Practices**

1. Use non-root users in containers
2. Drop unnecessary capabilities
3. Use read-only root filesystems where possible
4. Apply the principle of least privilege to ServiceAccounts
5. Use NetworkPolicies to restrict pod communication
6. Regularly audit and rotate secrets