The Ultimate Kubernetes Handbook

With Deeply Explained Code Comments

Your Personal AI Assistant

December 14, 2025

Contents

1 Core Concepts: The Foundation

1.1 Namespace (Virtual Clusters)

```yaml
apiVersion: v1
kind: Namespace
metadata:
  # 1. THE FOLDER NAME
  # Think of this as a folder on your computer.
  # Resources inside here are isolated from resources in other folders.
  # You delete the Namespace -> You delete everything inside it.
  name: production-env
```

Listing 1: Namespace Configuration

1.2 Pods (The Atomic Unit)

```yaml
apiVersion: v1
kind: Pod
metadata:
  # 1. THE POD'S ID CARD
  # Unique name for this specific object.
  # Used in commands like: 'kubectl delete pod my-first-pod'
  name: my-first-pod

  # 2. THE LOCATION
  # Places this Pod inside the folder we created above.
  namespace: production-env

  # 3. THE STICKY NOTE (Categorization)
  # Services use this label to find this Pod.
  # It is arbitrary: you could say 'tier: backend' or 'fruit: banana'.
  labels:
    app: demo-app
spec:
  containers:
  # 4. THE INTERNAL NICKNAME
  # Only used for logs: 'kubectl logs my-first-pod -c nginx-container'
  - name: nginx-container

    # 5. THE SOFTWARE PACKAGE
    # Downloads 'nginx' version 'alpine' from Docker Hub.
    image: nginx:alpine

    ports:
    # 6. THE EXPOSED PORT
    # This is purely informational. It tells humans "I listen on port 80".
    - containerPort: 80
```

Listing 2: Pod Configuration

1.3 Deployment (The Manager)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  # 1. THE MANAGER'S NAME
  # This is the name of the Deployment object itself.
  name: web-deployment
  namespace: production-env
spec:
  # 2. THE DESIRED TEAM SIZE
  # "I want exactly 3 clones of this pod running at all times."
  # If one dies, I will start a new one to keep this number at 3.
  replicas: 3

  # 3. THE SEARCH FILTER (Crucial!)
```

```
  # The Manager asks: "Which pods belong to me?"
  # Answer: "Any pod wearing the 'app: demo-app' badge."
  selector:
    matchLabels:
      app: demo-app  # <--- MUST MATCH Point #5 below

  # 4. THE BLUEPRINT (Cookie Cutter)
  # If I need to create a new Pod, I will use this template.
  template:
    metadata:
      # 5. THE BADGE
      # I will stick this label on every new Pod I create.
      # This ensures the Selector (Point #3) can find them.
      labels:
        app: demo-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

Listing 3: Deployment Configuration

2 Networking

2.1 Services (Internal Access)

```
apiVersion: v1
kind: Service
metadata:
  # 1. THE DNS NAME
  # Other pods can talk to this service by using this name.
  # Example URL: "http://web-service"
  name: web-service
  namespace: production-env
spec:
  # 2. THE TYPE OF ACCESS
  # ClusterIP = Internal Only (Default).
  # NodePort = Open a port on the server firewall.
  # LoadBalancer = Ask AWS/GCP for a Public IP.
  type: ClusterIP

  # 3. THE TARGET FINDER
  # "Send traffic to any Pod with this label."
  selector:
    app: demo-app

  ports:
    - protocol: TCP
      # 4. THE FRONT DOOR (Service Port)
      # This is the port other apps use to call the Service.
      port: 80

      # 5. THE BACK DOOR (Container Port)
      # This must match the port your container is actually listening on.
      targetPort: 80
```

Listing 4: Service Configuration

2.2 Ingress (External Routing)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: main-ingress
  namespace: production-env
  annotations:
    # 1. THE CONTROLLER INSTRUCTION
```

```
    # Specific configuration for Nginx (e.g., strip the path).
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  # 2. THE POLICE OFFICER
  # Tells the cluster which Ingress Controller should enforce these rules.
  ingressClassName: nginx

  rules:
  # 3. THE DOMAIN NAME
  # "I only care about traffic for myapp.com"
  - host: myapp.com
    http:
      paths:
      # 4. THE PATH RULE
      # "If the user visits myapp.com/api..."
      - path: /api
        pathType: Prefix
        backend:
          service:
            # 5. THE DESTINATION
            # "...send them to the backend-service on port 5000."
            name: backend-service
            port: { number: 5000 }
```

Listing 5: Ingress Rules

3 Storage

3.1 PersistentVolume (The Hardware)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  # 1. THE HARD DRIVE NAME
  # Managed by the IT Admin, not the Developer.
  name: pv-10gb
spec:
  # 2. THE SIZE
  # How big is this physical disk?
  capacity:
    storage: 10Gi

  # 3. THE ACCESS RULES
  # ReadWriteOnce = Only one node can mount this at a time (like a USB stick).
  # ReadWriteMany = Many nodes can share it (like a Shared Folder).
  accessModes:
    - ReadWriteOnce

  # 4. THE DRIVER
  # "hostPath" uses the Node's local disk (Testing only).
  # In production, use 'awsElasticBlockStore' or 'gcePersistentDisk'.
  hostPath:
    path: "/mnt/data"
```

Listing 6: PV Definition

3.2 PersistentVolumeClaim (The Ticket)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  # 1. THE TICKET NAME
  # The Pod will use this name to claim storage.
  name: db-storage-claim
  namespace: production-env
spec:
  # 2. THE REQUEST
```

```
  # "I need a drive that supports Single-Node writing."
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      # 3. THE SIZE NEEDED
      # "Find me a PV that has at least 5GB free."
      storage: 5Gi
```

Listing 7: PVC Request

4 Batch Jobs

4.1 CronJob (Scheduled)

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: nightly-backup
spec:
  # 1. THE SCHEDULE
  # Minute | Hour | Day-Month | Month | Day-Week
  # "Run at 00:00 (Midnight) every day."
  schedule: "0 0 * * *"

  # 2. THE JOB TEMPLATE
  # Every midnight, create a Job object with these settings.
  jobTemplate:
    spec:
      # 3. THE POD TEMPLATE
      # The Job will create a Pod with these settings.
      template:
        spec:
          containers:
          - name: backup
            image: backup-tool

          # 4. THE RESTART POLICY
          # "If the script crashes, try again (OnFailure)."
          # "Do not restart endlessly like a web server."
          restartPolicy: OnFailure
```

Listing 8: CronJob Example

5 Mini Project: Full Stack

5.1 Backend (Python API)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-api
spec:
  replicas: 2
  selector: { matchLabels: { app: backend } }
  template:
    metadata: { labels: { app: backend } }
    spec:
      containers:
      - name: python-api
        image: my-python-api:v1

        # 1. CONNECTING TO REDIS
        # We don't use IP addresses. We use the Service Name.
        env:
        - name: REDIS_HOST
```

```
          # This name 'redis-service' matches the Service YAML below.
          value: "redis-service"
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  # 1. INTERNAL ONLY
  # The public cannot access the API directly. Only the Frontend can.
  type: ClusterIP
  selector: { app: backend }
  ports: [{ port: 5000, targetPort: 5000 }]
```

Listing 9: Backend Deployment

5.2 Frontend (React)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-ui
spec:
  replicas: 3
  selector: { matchLabels: { app: frontend } }
  template:
    metadata: { labels: { app: frontend } }
    spec:
      containers:
      - name: react-ui
        image: my-react-app:v1

        # 1. CONNECTING TO BACKEND
        # The frontend browser needs to know where to send API requests.
        env:
        - name: API_URL
          value: "http://backend-service:5000"
---
apiVersion: v1
kind: Service
metadata:
  name: public-frontend
spec:
  # 1. PUBLIC ACCESS
  # This asks the Cloud Provider (AWS/GCP) for a Real Public IP.
  type: LoadBalancer
  selector: { app: frontend }
  ports:
    - port: 80          # Users hit port 80 (HTTP)
      targetPort: 80    # Container listens on 80
```

Listing 10: Frontend Public Access