🚀 Day 6: Kubernetes Learning Series

Advanced Workloads - Deployments, StatefulSets, DaemonSets & Jobs

📋 Understanding Workload Types

> 💡 **What are Workloads?**
> Workloads are higher-level abstractions that manage Pods for you. Instead of creating Pods directly, you use these controllers that handle Pod lifecycle, scaling, updates, and recovery automatically.

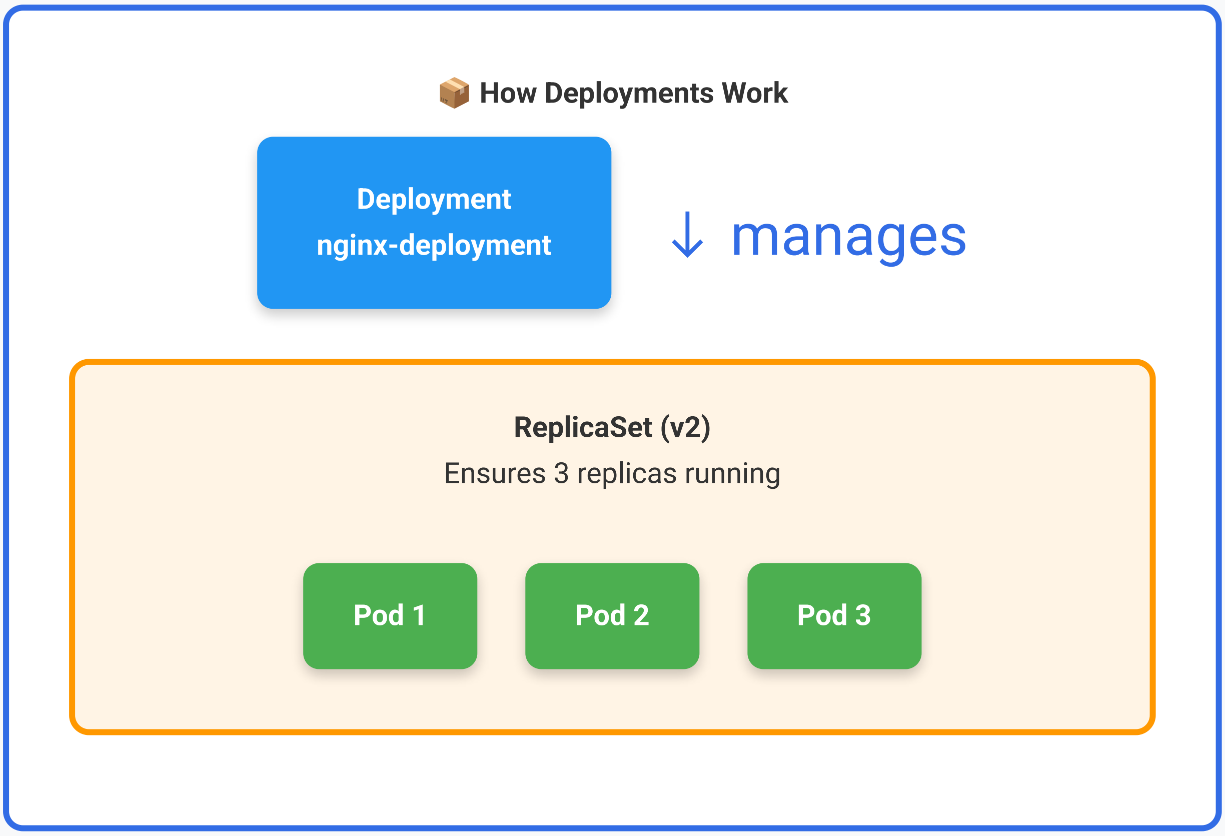
Quick Comparison

Workload Type	Purpose	When to Use
Deployment	Stateless applications with replicas	Web servers, APIs, microservices (MOST COMMON)
StatefulSet	Stateful applications needing persistent identity	Databases, message queues, clustered apps
DaemonSet	One pod per node	Monitoring agents, log collectors, node utilities
Job	Run once and complete	Batch processing, data migration, backups
CronJob	Scheduled periodic tasks	Scheduled backups, reports, cleanup tasks

🚢 Deployments

💡 **What is a Deployment?**
A Deployment is the **most commonly used** Kubernetes workload. It manages ReplicaSets, which manage Pods. It provides declarative updates, scaling, and rollback capabilities for stateless applications.

Deployment Architecture



📦 **How Deployments Work**

Why Use Deployments?

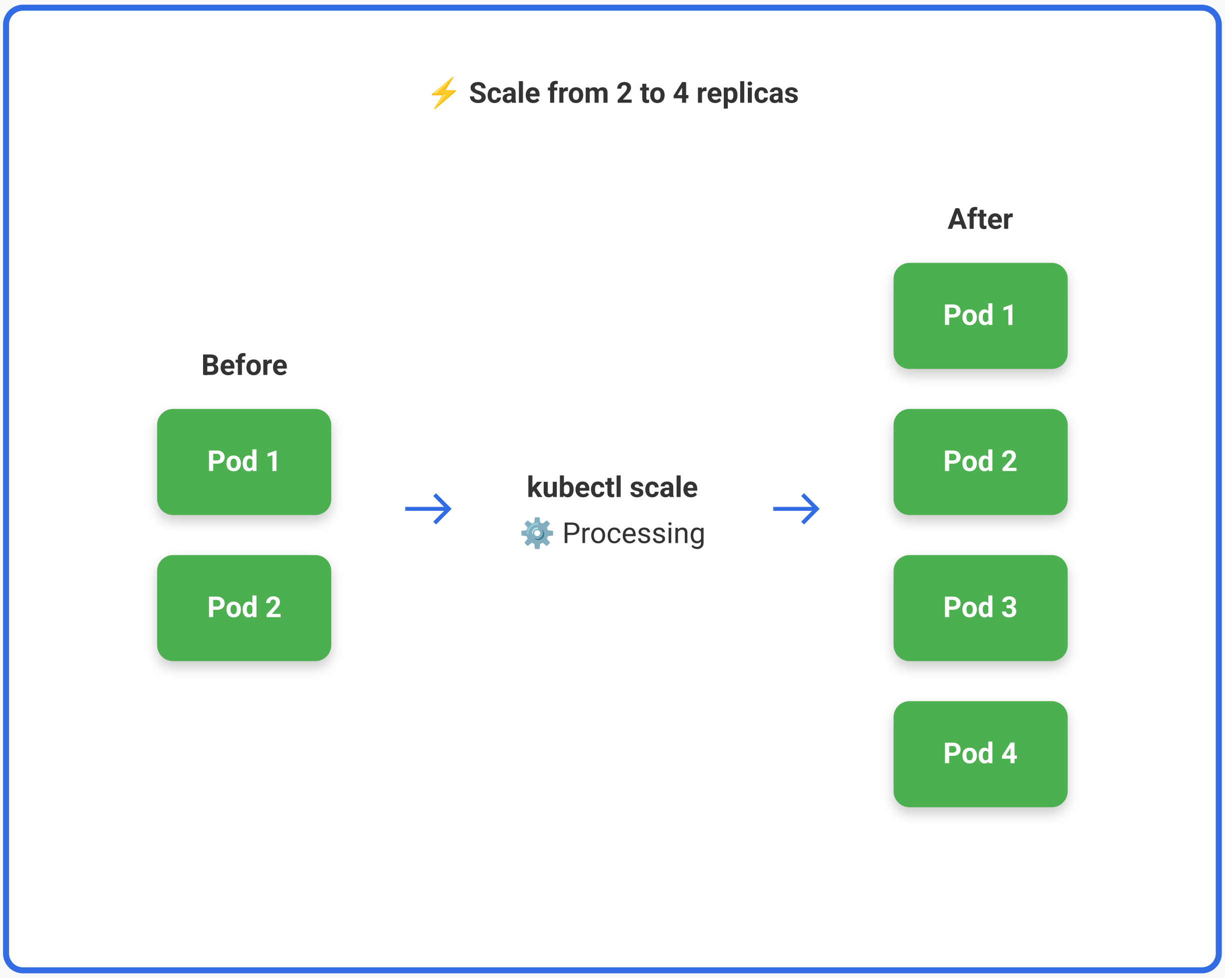
✅ **Self-Healing:** If a Pod dies, it automatically creates a new one
✅ **Scaling:** Easily scale up/down replicas
✅ **Rolling Updates:** Update without downtime
✅ **Rollback:** Revert to previous version if issues occur
✅ **Load Balancing:** Distributes traffic across all replicas

Basic Deployment Example

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3                    # Number of Pod copies
  selector:
    matchLabels:
      app: nginx               # Must match template labels
  template:                      # Pod template
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

Scaling Deployments

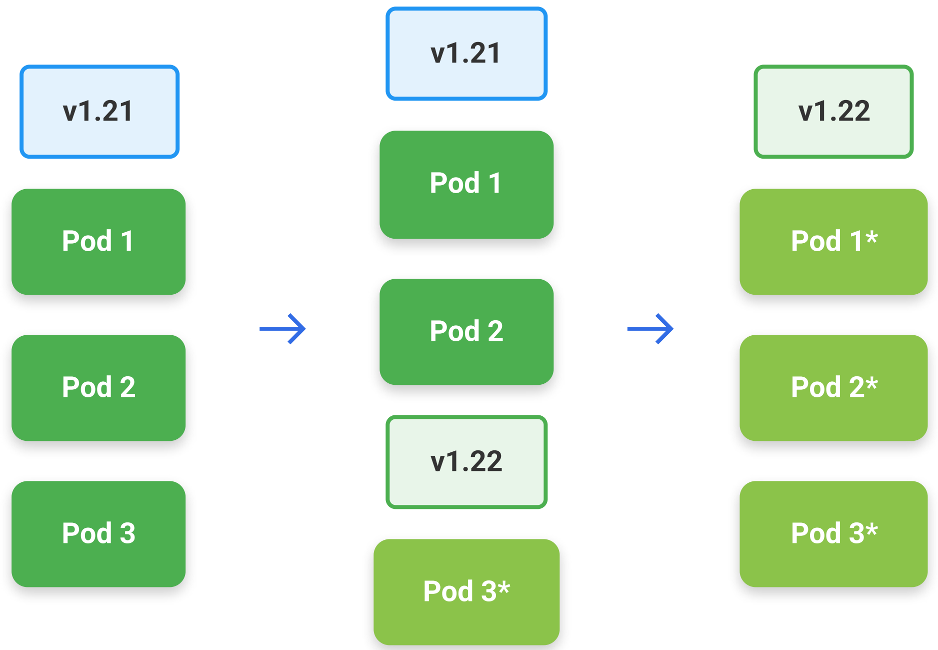


```
# Scale using kubectl command
kubectl scale deployment nginx-deployment --replicas=5

# Or update the YAML and apply
```

```yaml
spec:
  replicas: 5    # Change this value
```

Rolling Updates

🔄 Update from nginx:1.21 to nginx:1.22 with Zero Downtime



Gradually replaces old Pods with new ones

Deployment with Rolling Update Strategy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 5
  strategy:
    type: RollingUpdate          # Default strategy
    rollingUpdate:
      maxSurge: 1               # Max pods above desired count
      maxUnavailable: 1         # Max pods below desired count
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.22       # Updated version
        ports:
        - containerPort: 80
```

Rollback Example

```bash
# View rollout history
kubectl rollout history deployment nginx-deployment

# Rollback to previous version
kubectl rollout undo deployment nginx-deployment

# Rollback to specific revision
kubectl rollout undo deployment nginx-deployment --to-revision=2

# Check rollout status
kubectl rollout status deployment nginx-deployment
```

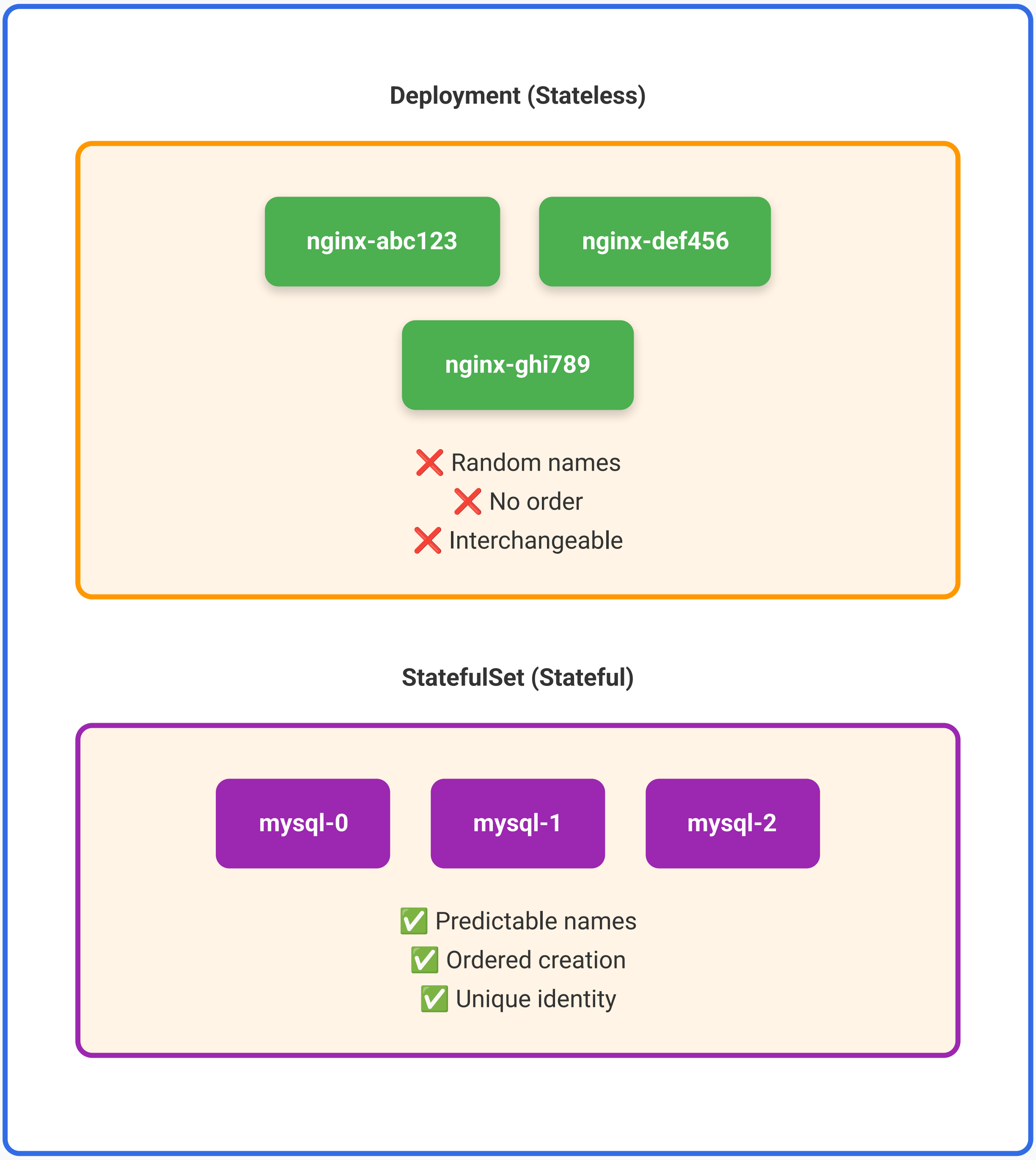
Common kubectl Commands for Deployments

```bash
# Create deployment
kubectl apply -f deployment.yaml

# List deployments
kubectl get deployments
kubectl get deploy

# Get detailed info
kubectl describe deployment nginx-deployment

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

# Update image
kubectl set image deployment/nginx-deployment nginx=nginx:1.22

# Delete deployment
kubectl delete deployment nginx-deployment
```

💾 StatefulSets

💡 **What is a StatefulSet?**
StatefulSets are for **stateful applications** that need stable network identities, persistent storage, and ordered deployment/scaling. Unlike Deployments, each Pod has a unique, persistent identity.

Deployment vs StatefulSet



Deployment (Stateless)

nginx-abc123
nginx-def456
nginx-ghi789

❌ Random names
❌ No order
❌ Interchangeable

StatefulSet (Stateful)

mysql-0
mysql-1
mysql-2

✅ Predictable names
✅ Ordered creation
✅ Unique identity

Key Features of StatefulSets

1. Stable Network Identity: Each Pod gets a persistent hostname (mysql-0, mysql-1, mysql-2)
2. Stable Storage: Each Pod has its own PersistentVolume that persists across restarts
3. Ordered Deployment: Pods created in order: 0, then 1, then 2
4. Ordered Termination: Pods deleted in reverse order: 2, then 1, then 0

5. **Ordered Updates:** Updates happen in reverse order with controlled rolling updates

StatefulSet Example - MySQL Cluster

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql-headless    # Headless service for stable DNS
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: password123
  volumeClaimTemplates:        # Creates PVC for each Pod
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 10Gi
```

Headless Service for StatefulSet

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-headless
spec:
  clusterIP: None              # Headless = no cluster IP
  selector:
    app: mysql
  ports:
```

```
  - port: 3306
    name: mysql
```

🔍 **DNS in StatefulSet:**
Each Pod gets a DNS entry: `mysql-0.mysql-headless.default.svc.cluster.local`
This allows other Pods to connect to specific StatefulSet Pods by name!

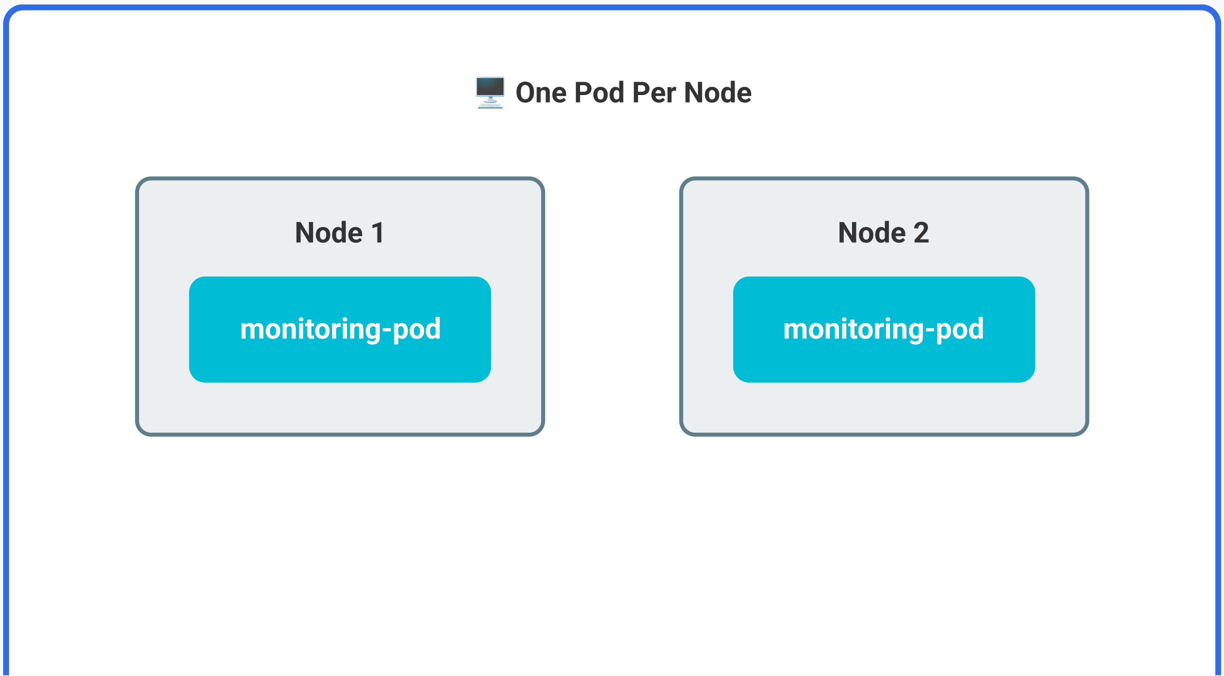
When to Use StatefulSets

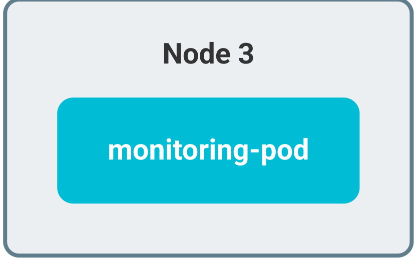
✅ **Databases:** MySQL, PostgreSQL, MongoDB
✅ **Message Queues:** Kafka, RabbitMQ
✅ **Distributed Systems:** Zookeeper, etcd, Cassandra
✅ **Any app requiring:** Stable storage, network identity, or ordered operations

🔄 DaemonSets

💡 **What is a DaemonSet?**
A DaemonSet ensures that **one copy of a Pod runs on every node** (or selected nodes) in the cluster. Perfect for node-level services like monitoring and logging.

How DaemonSets Work





✨ New node joins → Automatically gets a Pod
🗑️ Node removed → Pod automatically deleted

Common Use Cases

📊 **Monitoring Agents:** Prometheus Node Exporter, Datadog agent
📝 **Log Collectors:** Fluentd, Logstash, Filebeat
🔒 **Security Agents:** Antivirus, intrusion detection
🌐 **Network Plugins:** CNI plugins like Calico, Flannel
💾 **Storage Daemons:** Ceph, GlusterFS agents

DaemonSet Example - Log Collector

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    app: fluentd
spec:
  selector:
    matchLabels:
      app: fluentd
  template:
    metadata:
      labels:
        app: fluentd
    spec:
      containers:
      - name: fluentd
        image: fluentd:v1.14
        volumeMounts:
        - name: varlog
          mountPath: /var/log      # Access node logs
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
```

4. Job - Database Migration

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: db-migration
spec:
  template:
    spec:
      containers:
      - name: migrate
        image: migrate/migrate
        command:
        - migrate
        - -path
        - /migrations
        - -database
        - mysql://root:password@mysql-0.mysql:3306/mydb
        - up
      restartPolicy: Never
  backoffLimit: 3
```

5. CronJob - Nightly Backup

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: nightly-backup
spec:
  schedule: "0 2 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: mysql:8.0
            command:
            - /bin/sh
            - -c
            - mysqldump -h mysql-0.mysql -u root -ppassword mydb > /backup/
            volumeMounts:
            - name: backup
              mountPath: /backup
          volumes:
          - name: backup
            persistentVolumeClaim:
              claimName: backup-pvc
          restartPolicy: OnFailure
```

🔧 Common Issues & Troubleshooting

Deployment Issues

Problem: Pods not starting after update
Solutions:

1. Check rollout status: `kubectl rollout status deployment/myapp`
2. Check events: `kubectl describe deployment myapp`
3. Check pod logs: `kubectl logs deployment/myapp`
4. Rollback if needed: `kubectl rollout undo deployment/myapp`

Problem: Image pull errors
Solutions:

1. Verify image name and tag are correct
2. Check imagePullSecrets if using private registry
3. Use `kubectl describe pod` to see detailed error

StatefulSet Issues

Problem: Pods stuck in Pending
Solutions:

1. Check PVC status: `kubectl get pvc`
2. Verify storage class exists: `kubectl get storageclass`
3. Check if PersistentVolumes are available
4. Review events: `kubectl describe statefulset mysql`

Problem: StatefulSet not scaling down
Solutions:

1. StatefulSets scale down in reverse order (highest first)
2. Ensure pods are not in error state
3. Check for PodDisruptionBudgets that may prevent deletion

DaemonSet Issues

Problem: DaemonSet not running on all nodes

Solutions:
1. Check node selectors and affinity rules
2. Check taints on nodes: `kubectl describe node`
3. Add tolerations if nodes are tainted
4. Verify nodes are Ready: `kubectl get nodes`

Job/CronJob Issues

Problem: Job keeps restarting
Solutions:
1. Check logs: `kubectl logs job/myjob`
2. Verify restartPolicy is correct (Never or OnFailure)
3. Check if backoffLimit is reached
4. Ensure the job's command exits with code 0 on success

Problem: CronJob not running at scheduled time
Solutions:
1. Verify cron syntax is correct
2. Check if CronJob is suspended: `kubectl get cronjob`
3. Verify startingDeadlineSeconds isn't too restrictive
4. Check timezone considerations (CronJobs use UTC by default)
5. Look at recent jobs: `kubectl get jobs`

📚 Quick Reference - kubectl Commands

Deployments

```
# Create/Update
kubectl apply -f deployment.yaml
kubectl create deployment nginx --image=nginx:1.21

# List and view
kubectl get deployments
kubectl get deploy nginx -o yaml
kubectl describe deployment nginx

# Scale
```

```
kubectl scale deployment nginx --replicas=5
kubectl autoscale deployment nginx --min=3 --max=10 --cpu-percent=80

# Update
kubectl set image deployment/nginx nginx=nginx:1.22
kubectl edit deployment nginx

# Rollout management
kubectl rollout status deployment/nginx
kubectl rollout history deployment/nginx
kubectl rollout undo deployment/nginx
kubectl rollout restart deployment/nginx

# Delete
kubectl delete deployment nginx
```

StatefulSets

```
# Create/Update
kubectl apply -f statefulset.yaml

# List and view
kubectl get statefulsets
kubectl get sts mysql -o wide
kubectl describe statefulset mysql

# Scale
kubectl scale statefulset mysql --replicas=5

# Check PVCs
kubectl get pvc

# Delete (keeps PVCs by default)
kubectl delete statefulset mysql

# Delete including PVCs
kubectl delete statefulset mysql
kubectl delete pvc data-mysql-0 data-mysql-1 data-mysql-2
```

DaemonSets

```
# Create/Update
kubectl apply -f daemonset.yaml

# List and view
kubectl get daemonsets
kubectl get ds fluentd -o wide
kubectl describe daemonset fluentd

# Check which nodes have pods
kubectl get pods -o wide -l app=fluentd
```

```
# Update
kubectl set image daemonset/fluentd fluentd=fluentd:v1.15

# Delete
kubectl delete daemonset fluentd
```

Jobs

```
# Create
kubectl apply -f job.yaml
kubectl create job test --image=busybox -- echo "Hello"

# List and view
kubectl get jobs
kubectl describe job myjob

# View logs
kubectl logs job/myjob
kubectl logs -f job/myjob

# Delete
kubectl delete job myjob

# Delete completed jobs
kubectl delete jobs --field-selector status.successful=1
```

CronJobs

```
# Create
kubectl apply -f cronjob.yaml
kubectl create cronjob test --image=busybox --schedule="*/5 * * * *" -- ech

# List and view
kubectl get cronjobs
kubectl get cj backup -o wide
kubectl describe cronjob backup

# Manually trigger
kubectl create job manual-run --from=cronjob/backup

# Suspend/Resume
kubectl patch cronjob backup -p '{"spec":{"suspend":true}}'
kubectl patch cronjob backup -p '{"spec":{"suspend":false}}'

# View recent jobs
kubectl get jobs --sort-by=.metadata.creationTimestamp
```

```
# Delete
kubectl delete cronjob backup
```

💪 Practice Exercises

Exercise 1: Deployment with Rolling Update

1. Create a Deployment with nginx:1.19 and 3 replicas
2. Expose it with a Service
3. Update the image to nginx:1.21
4. Watch the rollout process
5. Rollback to previous version
6. Scale to 5 replicas

Exercise 2: StatefulSet Database

1. Create a StatefulSet for PostgreSQL with 3 replicas
2. Create a Headless Service for it
3. Verify each pod has its own PVC
4. Connect to each pod and verify unique identity
5. Scale down to 2 replicas and observe behavior

Exercise 3: DaemonSet for Monitoring

1. Create a DaemonSet that runs on all nodes
2. Use a simple busybox container that logs node info
3. Verify one pod per node exists
4. Update to run only on nodes with label env=production
5. Label one node and verify DaemonSet behavior

Exercise 4: Jobs and CronJobs

1. Create a Job that runs 5 times with parallelism of 2
2. Watch the pods being created in batches
3. Create a CronJob that runs every 2 minutes
4. Manually trigger the CronJob
5. Suspend the CronJob
6. Clean up old completed jobs

Exercise 5: Complete Application Stack

1. Deploy a web app using Deployment (3 replicas)
2. Deploy MySQL using StatefulSet
3. Deploy log collector using DaemonSet
4. Create a Job for initial database setup
5. Create a CronJob for daily backups
6. Test the entire stack end-to-end

🎓 Key Takeaways - Day 6

Deployments

🚢 Most common workload for stateless apps
🔄 Rolling updates with zero downtime
↩️ Easy rollback if issues occur
📈 Simple scaling up and down

StatefulSets

💾 For stateful applications needing persistence

- 🔢 Stable, ordered pod identity
- 💿 Each pod gets its own persistent storage
- 🗂️ Perfect for databases and clustered apps

DaemonSets

- 🔄 One pod per node automatically
- 📊 Ideal for monitoring and logging
- 🖥️ Runs on every node in cluster
- 🎯 Use node selectors for targeted placement

Jobs & CronJobs

- ⚡ Jobs for one-time batch tasks
- ⏰ CronJobs for scheduled recurring tasks
- ✅ Run to completion, then stop
- 🔁 Perfect for backups, reports, migrations

Choosing the Right Workload

- 🌐 Web apps, APIs → **Deployment**
- 🗄️ Databases → **StatefulSet**
- 📝 Node monitoring → **DaemonSet**
- 📦 Data migration → **Job**
- 🕐 Scheduled backups → **CronJob**

🌍 Real-World Scenarios

Scenario 1: E-Commerce Website

Requirements: Web frontend, product API, shopping cart, database, image processing

🚢 **Deployment:** Web Frontend (5 replicas)
- Handles user traffic
- Stateless, can scale horizontally
- Rolling updates for new features

🚢 **Deployment:** Product API (3 replicas)
- REST API for product catalog
- Stateless microservice

💾 **StatefulSet:** PostgreSQL Database (1 master, 2 replicas)
- Persistent storage for orders
- Each pod needs stable identity
- PersistentVolumes for data

⚡ **Job:** Initial Data Migration
- Import existing product catalog
- Runs once at deployment

⏰ **CronJob:** Nightly Reports (0 3 * * *)
- Generate sales reports at 3 AM
- Email to management

⏰ **CronJob:** Cart Cleanup (0 */6 * * *)
- Remove abandoned carts every 6 hours

🔄 **DaemonSet:** Fluentd Log Collection
- Collects logs from all nodes
- Sends to centralized logging

Scenario 2: Data Processing Pipeline

Requirements: Process large datasets, ML model training, monitoring

⚡ **Job:** Data Ingestion
- Import CSV files from S3
- Parallelism: 10 (process 10 files at once)
- Completions: 100 (total files)

⚡ **Job:** Data Transformation
- Clean and transform data
- Runs after ingestion completes

⚡ **Job:** ML Model Training

- Train models on processed data
- GPU nodes with node affinity
- Long-running (activeDeadlineSeconds: 86400)

🚢 **Deployment:** Model Serving API (3 replicas)

- Serve predictions via REST API
- Load trained model from storage

⏰ **CronJob:** Daily Retraining (0 1 * * *)

- Retrain models with new data
- Runs at 1 AM daily

🔄 **DaemonSet:** Prometheus Node Exporter

- Monitor resource usage on all nodes

Scenario 3: SaaS Multi-Tenant Platform

Requirements: Web app, API gateway, tenant databases, background processing

🚢 **Deployment:** Web Application (10 replicas)

- Multi-tenant SaaS frontend
- Auto-scaling based on traffic

🚢 **Deployment:** API Gateway (5 replicas)

- Routes requests to microservices
- Rate limiting per tenant

💾 **StatefulSet:** Redis Cache Cluster (3 nodes)

- Session storage and caching
- Master-slave replication

💾 **StatefulSet:** MongoDB Cluster (3 nodes per tenant)

- Database isolation per tenant
- Replica sets for HA

🚢 **Deployment:** Background Workers (5 replicas)

- Process async tasks
- Email sending, notifications

⏰ **CronJob:** Usage Metrics (0 0 * * *)

- Calculate tenant usage daily
- Update billing records

⏰ **CronJob:** Database Backup (0 2 * * *)

- Backup all tenant databases
- Upload to S3

🔄 **DaemonSet:** Security Agent
- Monitor for security threats
- Run on all nodes

💡 Advanced Tips & Tricks

Deployment Strategies

Blue-Green Deployment Pattern:
1. Create new Deployment (green) alongside old (blue)
2. Test green deployment thoroughly
3. Switch Service selector to green
4. Keep blue for quick rollback if needed

Canary Deployment Pattern:
1. Deploy new version with 1 replica
2. Monitor metrics and errors
3. Gradually increase replicas (10%, 25%, 50%, 100%)
4. Rollback immediately if issues detected

Resource Management

Always Set Resource Requests & Limits:

```
resources:
  requests:
    memory: "256Mi"      # Minimum guaranteed
    cpu: "100m"          # 0.1 CPU
  limits:
    memory: "512Mi"      # Maximum allowed
    cpu: "500m"          # 0.5 CPU
```

✅ Prevents resource starvation
✅ Enables proper scheduling

✅ Protects against memory leaks

Health Checks

Liveness Probe: Restart container if unhealthy
Readiness Probe: Stop sending traffic if not ready
Startup Probe: Wait for slow-starting containers

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 30
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 5
```

Pod Disruption Budgets

Protect critical applications during updates:

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: webapp-pdb
spec:
  minAvailable: 2         # Always keep 2 pods running
  selector:
    matchLabels:
      app: webapp
```

✅ Prevents all pods from being terminated at once
✅ Ensures high availability during maintenance

StatefulSet Gotchas

⚠️ **Important Things to Know:**

1. **PVCs are NOT deleted** when StatefulSet is deleted
   → Must manually delete PVCs to free storage

2. **Pods are created sequentially** by default
   → Pod-1 waits for Pod-0 to be ready
   → Use `podManagementPolicy: Parallel` for faster startup

3. **No automatic rolling restart** on ConfigMap/Secret changes
   → Must manually restart pods

4. **Headless Service is required**
   → Provides stable DNS for each pod

Job Performance Tips

Optimize Parallel Processing:

```
spec:
  completions: 100         # Total work items
  parallelism: 10          # Concurrent pods
  backoffLimit: 3          # Retry failed pods
  activeDeadlineSeconds: 600  # Timeout after 10 min
  ttlSecondsAfterFinished: 3600  # Clean up after 1 hour
```

💡 **Tip:** Set parallelism to match available nodes for best performance
💡 **Tip:** Use ttlSecondsAfterFinished to auto-clean completed jobs

CronJob Best Practices

Make Jobs Idempotent:
→ Jobs should produce same result if run multiple times
→ CronJobs can trigger multiple times if cluster is busy

Handle Missed Schedules:

```
spec:
  startingDeadlineSeconds: 300  # Skip if >5min late
  concurrencyPolicy: Forbid     # Don't overlap runs
```

Test Cron Expressions:

→ Use crontab.guru to validate schedules
→ Remember: Kubernetes uses UTC timezone by default

🔍 Monitoring & Debugging

Essential Monitoring Commands

```
# Watch resources in real-time
kubectl get pods -w
kubectl get deployments -w

# Top resource consumers
kubectl top nodes
kubectl top pods
kubectl top pods --containers

# Get pod events
kubectl get events --sort-by='.lastTimestamp'
kubectl get events --field-selector involvedObject.name=mypod

# Check resource usage
kubectl describe node mynode
kubectl describe pod mypod

# View logs from all containers
kubectl logs -f deployment/myapp --all-containers=true
kubectl logs -f statefulset/mysql --prefix=true
```

Debug Failing Deployments

```
# Check rollout status
kubectl rollout status deployment/myapp

# View rollout history
kubectl rollout history deployment/myapp

# Check why pods aren't starting
kubectl describe deployment myapp
kubectl describe replicaset myapp-xxxxx
kubectl describe pod myapp-xxxxx

# Get pod logs (even if crashed)
kubectl logs myapp-xxxxx --previous
```

```
# Exec into running pod
kubectl exec -it myapp-xxxxx -- /bin/bash
```

Debug StatefulSet Issues

```
# Check PVC status
kubectl get pvc
kubectl describe pvc data-mysql-0

# Check PV status
kubectl get pv

# View StatefulSet events
kubectl describe statefulset mysql

# Check if Headless Service exists
kubectl get svc mysql -o yaml

# Test DNS resolution
kubectl run -it debug --image=busybox --rm -- nslookup mysql-0.mysql
```

Debug Jobs & CronJobs

```
# List all jobs from cronjob
kubectl get jobs --selector=job-name=mycronjob

# Get pods from job
kubectl get pods --selector=job-name=myjob

# View job logs
kubectl logs job/myjob

# Check why job failed
kubectl describe job myjob

# View cronjob schedule
kubectl get cronjob mycronjob -o yaml

# Check last execution time
kubectl describe cronjob mycronjob
```

🚀 What's Next?

Day 7 Preview: Persistent Storage & Volumes

Get ready to master Kubernetes storage! You'll learn:

💾 **Volume Basics:** Understanding ephemeral vs persistent storage
→ emptyDir, hostPath, and volume types
→ When to use each volume type

📦 **PersistentVolumes (PV):** Cluster-level storage resources
→ Creating and managing PVs
→ Access modes and reclaim policies

📋 **PersistentVolumeClaims (PVC):** Requesting storage
→ Binding PVCs to PVs
→ Using PVCs in Pods

🏪 **StorageClasses:** Dynamic provisioning
→ Automatic PV creation
→ Cloud provider integration

☁️ **Cloud Storage:** AWS EBS, GCP PD, Azure Disk
→ Using cloud storage in Kubernetes
→ Volume snapshots and backups

📊 **Storage Best Practices:** Production-ready patterns
→ Backup strategies
→ Performance optimization

Congratulations! 🎉

You now understand all major Kubernetes workload types!

You can now:
✅ Deploy stateless applications with Deployments
✅ Perform rolling updates with zero downtime
✅ Run stateful applications with StatefulSets
✅ Deploy node-level services with DaemonSets
✅ Run batch jobs and scheduled tasks
✅ Choose the right workload for any scenario
✅ Debug and troubleshoot workload issues
✅ Implement real-world application architectures