**KUBERNETES CASE STUDIES WITH REAL-WORLD SCENARIOS, COMMANDS, AND SOLUTIONS**

**Case Study 1: High Availability Microservices Deployment**

**Scenario:**

A growing **e-commerce startup** faces frequent website crashes during peak traffic. They need a scalable solution.

**Solution:**

Deploy **Product & Cart services** with Kubernetes  
Implement **Horizontal Pod Autoscaler (HPA)** to scale dynamically

**Step-by-Step Guide**

1️. **Deploy Product Service (with 3 replicas)**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: product-service**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: product**

**template:**

**metadata:**

**labels:**

**app: product**

**spec:**

**containers:**

**- name: product**

**image: myregistry/product-service:v1**

**ports:**

**- containerPort: 8080**

**kubectl apply -f product-service.yaml**

**2️. Expose Product Service**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: product-service**

**spec:**

**selector:**

**app: product**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 8080**

**type: LoadBalancer**

**kubectl apply -f product-service-service.yaml**

**3️. Enable Horizontal Pod Autoscaler (HPA)**

**kubectl autoscale deployment product-service --cpu-percent=50 --min=3 --max=10**

**4️. Test Scaling Under Load**

**kubectl get hpa**

**Outcome: The e-commerce website scales automatically during traffic spikes.**

**Case Study 2: Kubernetes Disaster Recovery with Backups**

**Scenario:**

**A SaaS company accidentally deletes a production namespace. They need a disaster recovery solution.**

**Solution:**

**Install Velero for automated backups  
Restore deleted Kubernetes namespace**

**Step-by-Step Guide**

**1️. Install Velero for Backup**

**velero install --provider aws --plugins velero/velero-plugin-for-aws:v1.2.0 --bucket mybackup --backup-location-config region=us-east-1**

**2️. Create a Backup**

**velero backup create my-backup --include-namespaces=my-namespace**

**3️. Simulate Data Loss (Delete Namespace)**

**kubectl delete namespace my-namespace**

**4️. Restore Data from Backup**

**velero restore create --from-backup my-backup**

**Outcome: The lost data is fully recovered in minutes.**

**Case Study 3: Securing Kubernetes Workloads**

**Scenario:**

**A FinTech company needs to secure workloads from unauthorized access.**

**Solution:**

**Use Network Policies to limit database access  
Implement Role-Based Access Control (RBAC)**

**Step-by-Step Guide**

**1️. Restrict Database Access with Network Policies**

**apiVersion: networking.k8s.io/v1**

**kind: NetworkPolicy**

**metadata:**

**name: db-network-policy**

**spec:**

**podSelector:**

**matchLabels:**

**app: database**

**policyTypes:**

**- Ingress**

**ingress:**

**- from:**

**- podSelector:**

**matchLabels:**

**app: backend**

**kubectl apply -f db-network-policy.yaml**

**2️. Restrict User Permissions Using RBAC**

**apiVersion: rbac.authorization.k8s.io/v1**

**kind: Role**

**metadata:**

**namespace: default**

**name: developer-role**

**rules:**

**- apiGroups: [""]**

**resources: ["pods"]**

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

**yaml**

**CopyEdit**

**apiVersion: rbac.authorization.k8s.io/v1**

**kind: RoleBinding**

**metadata:**

**name: developer-binding**

**subjects:**

**- kind: User**

**name: developer**

**roleRef:**

**kind: Role**

**name: developer-role**

**apiGroup: rbac.authorization.k8s.io**

**kubectl apply -f developer-role.yaml**

**kubectl apply -f developer-rolebinding.yaml**

**Outcome: Secure Kubernetes workloads & meet compliance.**

**Case Study 4: Zero Downtime Deployment with Blue-Green Strategy**

**Scenario:**

**A media streaming platform experiences downtime during software updates.**

**Solution:**

**Use Blue-Green Deployment for zero-downtime updates**

**Step-by-Step Guide**

**1️. Deploy Current (Blue) Version**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: app-blue**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: app-blue**

**template:**

**metadata:**

**labels:**

**app: app-blue**

**spec:**

**containers:**

**- name: app**

**image: myregistry/app:v1**

**ports:**

**- containerPort: 8080**

**kubectl apply -f app-blue.yaml**

**2️. Deploy New (Green) Version**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: app-green**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: app-green**

**template:**

**metadata:**

**labels:**

**app: app-green**

**spec:**

**containers:**

**- name: app**

**image: myregistry/app:v2**

**ports:**

**- containerPort: 8080**

**kubectl apply -f app-green.yaml**

**3️. Switch Traffic to Green (New Version)**

**kubectl apply -f app-service.yaml**

**Outcome: No downtime during deployments.**

**Case Study 5: Kubernetes Cost Optimization with Cluster Autoscaler**

**Scenario:**

**A SaaS startup wants to reduce Kubernetes cloud costs.**

**Solution:**

**Use Cluster Autoscaler to optimize resources**

**Step-by-Step Guide**

**1️. Install Cluster Autoscaler**

**kubectl apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/master/cluster-autoscaler/cloud-provider.yaml**

**2️. Enable Autoscaling for Node Pool**

**gcloud container clusters update my-cluster --enable-autoscaling --min-nodes=1 --max-nodes=5**

**3️. Monitor Autoscaler**

**kubectl get hpa**

**Outcome: Reduced cloud costs by 30%.**

**Case Study 6: Multi-Tenant Kubernetes Security**

**Scenario:**

**A cloud service provider wants to isolate customer workloads in a shared cluster.**

**Solution:**

**Use Namespaces & Network Policies for multi-tenancy**

**Step-by-Step Guide**

**1️. Create Namespaces for Tenants**

**kubectl create namespace tenant-a**

**kubectl create namespace tenant-b**

**2️. Deploy Apps in Namespaces**

**kubectl apply -f tenant-a-app.yaml**

**3️. Restrict Access Between Tenants**

**apiVersion: networking.k8s.io/v1**

**kind: NetworkPolicy**

**metadata:**

**name: tenant-a-restrict**

**namespace: tenant-a**

**spec:**

**podSelector:**

**matchLabels:**

**app: tenant-a-app**

**policyTypes:**

**- Ingress**

**ingress:**

**- from:**

**- podSelector:**

**matchLabels:**

**app: tenant-a-app**

**kubectl apply -f tenant-a-restrict.yaml**

**Outcome: Customers cannot access each other’s workloads.**