Perfect—you’ve already done a **basic Kubernetes project** (MongoDB + web hosting), so now we need to **level up** to an **admin-grade, complex project** that showcases **real-world production challenges**.

## **🚀 Project: Kubernetes Production-Grade E-commerce App (Admin Focus)**

This will mimic what Kubernetes administrators do in real companies.

We’ll make it **local** (using Minikube or KIND), but designed to be **cloud-ready**.

## **🛠️ Project Components**

1. **Microservices** (5 services):
   1. frontend (React/Angular)
   2. product-service (Go/Node.js)
   3. order-service (Python/Flask)
   4. auth-service (JWT-based)
   5. mongo-db (Database)
2. **Admin-level K8s features:**
   1. **Ingress Controller + SSL (cert-manager)**
   2. **Horizontal Pod Autoscaling (HPA)**
   3. **Pod Disruption Budgets (PDB)**
   4. **Resource Quotas and Limits**
   5. **Secrets and ConfigMaps management**
   6. **Network Policies for service isolation**
   7. **Centralized logging (EFK)**
   8. **Monitoring (Prometheus + Grafana)**
   9. **Helm charts for deployments**
   10. **CI/CD pipeline with rollback strategy**
3. **Advanced scenarios:**
   1. Simulate node failure → auto rescheduling
   2. Scale workers dynamically
   3. Perform rolling updates and canary deployments
   4. Backup and restore MongoDB
   5. Pod Security Policies (PSP) / Pod Security Admission

## **📝 Step-by-Step Execution**

### **Step 1: Setup Local Multi-node Cluster**

* Use **KIND (Kubernetes-in-Docker)** for multiple nodes:

cat <<EOF | kind create cluster --name ecommerce-cluster --config=-  
kind: Cluster  
apiVersion: kind.x-k8s.io/v1alpha4  
nodes:  
- role: control-plane  
- role: worker  
- role: worker  
EOF  
kubectl get nodes

### **Step 2: Deploy Ingress + SSL**

* Install NGINX Ingress Controller:

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml>

* Install cert-manager for auto SSL:

kubectl apply -f <https://github.com/jetstack/cert-manager/releases/download/v1.12.0/cert-manager.yaml>

### **Step 3: Database Layer (MongoDB)**

* Create Persistent Volume (PV) & Persistent Volume Claim (PVC).
* Deploy MongoDB StatefulSet with replica count = 3 for HA.
* Expose as a ClusterIP service.

### **Step 4: Deploy Backend Microservices**

* Create separate namespaces (auth, product, order).
* Package each service as a **Helm chart**.
* Add resource limits and liveness/readiness probes.

### **Step 5: Frontend Deployment**

* Create frontend service (React/Angular).
* Expose via Ingress (with SSL cert from cert-manager).

### **Step 6: Implement HPA and PDB**

* For each service:

kubectl autoscale deployment product-deployment --cpu-percent=50 --min=1 --max=10

* Add Pod Disruption Budgets for high availability.

### **Step 7: Secure Networking**

* Create **NetworkPolicies** to:
  + Allow frontend → backend
  + Deny cross-namespace access

### **Step 8: Logging and Monitoring**

* Deploy:
  + **EFK Stack** (Elasticsearch, Fluentd, Kibana)
  + **Prometheus + Grafana**
* Visualize CPU, memory, pod restarts, MongoDB health.

### **Step 9: CI/CD + Rolling Updates**

* Setup GitHub Actions or Jenkins pipeline:
  + Build Docker image → push to local registry
  + Deploy Helm chart with rolling updates
* Test Canary deployment for product service.

### **Step 10: Disaster Recovery**

* Take etcd backup and restore cluster.
* Backup and restore MongoDB data using mongodump.

## **🎯 Learning Outcomes**

* Kubernetes multi-namespace, multi-service architecture
* Cluster security (RBAC, NetworkPolicies)
* Scaling and autoscaling services
* Monitoring and logging like a real production setup
* Admin-level troubleshooting (node failures, pod evictions)
* Helm and CI/CD best practices
* Disaster recovery planning

Would you like me to prepare a **full folder structure + YAML/Helm manifests** for this project so you can directly start implementing it step by step?

========================================================================

Great choice! Here’s a **ready-to-use folder structure** with explanations and **step-by-step YAML/Helm manifests** for the Kubernetes Production-Grade E-commerce App.

## **📂 Folder Structure**

k8s-ecommerce-project/  
│── README.md  
│── kind-cluster.yaml # KIND multi-node config  
│── ingress/  
│ ├── ingress-controller.yaml  
│ ├── ingress-rules.yaml  
│── cert-manager/  
│ ├── cert-manager.yaml  
│ ├── cluster-issuer.yaml  
│── monitoring/  
│ ├── prometheus.yaml  
│ ├── grafana.yaml  
│── logging/  
│ ├── elasticsearch.yaml  
│ ├── fluentd.yaml  
│ ├── kibana.yaml  
│── mongodb/  
│ ├── pv-pvc.yaml  
│ ├── statefulset.yaml  
│ ├── service.yaml  
│── helm-charts/  
│ ├── auth-service/  
│ ├── product-service/  
│ ├── order-service/  
│ ├── frontend/  
│── network-policies/  
│ ├── frontend-policy.yaml  
│ ├── backend-policy.yaml  
│── hpa-pdb/  
│ ├── hpa.yaml  
│ ├── pdb.yaml  
│── ci-cd/  
│ ├── jenkinsfile  
│ ├── github-actions.yaml  
│── disaster-recovery/  
│ ├── etcd-backup.md  
│ ├── mongo-backup.md

## **1️⃣ KIND Multi-Node Cluster**

kind-cluster.yaml

kind: Cluster  
apiVersion: kind.x-k8s.io/v1alpha4  
nodes:  
- role: control-plane  
- role: worker  
- role: worker

**Create cluster:**

kind create cluster --name ecommerce-cluster --config=kind-cluster.yaml  
kubectl get nodes

## **2️⃣ Ingress Controller**

ingress/ingress-controller.yaml

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml>

## **3️⃣ Cert-Manager & SSL**

cert-manager/cert-manager.yaml

kubectl apply -f <https://github.com/jetstack/cert-manager/releases/download/v1.12.0/cert-manager.yaml>

cert-manager/cluster-issuer.yaml

apiVersion: cert-manager.io/v1  
kind: ClusterIssuer  
metadata:  
 name: letsencrypt-prod  
spec:  
 acme:  
 server: <https://acme-v02.api.letsencrypt.org/directory> email: [admin@example.com](mailto:admin@example.com) privateKeySecretRef:  
 name: letsencrypt-prod  
 solvers:  
 - http01:  
 ingress:  
 class: nginx  
 kubectl apply -f cert-manager/cluster-issuer.yaml

## **4️⃣ MongoDB StatefulSet**

mongodb/pv-pvc.yaml

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: mongodb-pvc  
spec:  
 accessModes: [ReadWriteOnce]  
 resources:  
 requests:  
 storage: 5Gi

mongodb/statefulset.yaml

apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
 name: mongodb  
spec:  
 serviceName: "mongodb"  
 replicas: 3  
 selector:  
 matchLabels:  
 app: mongodb  
 template:  
 metadata:  
 labels:  
 app: mongodb  
 spec:  
 containers:  
 - name: mongo  
 image: mongo:6  
 ports:  
 - containerPort: 27017  
 volumeMounts:  
 - name: mongo-storage  
 mountPath: /data/db  
 volumeClaimTemplates:  
 - metadata:  
 name: mongo-storage  
 spec:  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 5Gi

## **5️⃣ Helm Charts for Microservices**

Example: helm-charts/product-service/values.yaml

replicaCount: 2  
image:  
 repository: product-service  
 tag: latest  
resources:  
 requests:  
 cpu: 100m  
 memory: 256Mi  
 limits:  
 cpu: 500m  
 memory: 512Mi

## **6️⃣ HPA & PDB**

hpa-pdb/hpa.yaml

apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
 name: product-hpa  
spec:  
 scaleTargetRef:  
 apiVersion: apps/v1  
 kind: Deployment  
 name: product-service  
 minReplicas: 2  
 maxReplicas: 10  
 metrics:  
 - type: Resource  
 resource:  
 name: cpu  
 target:  
 type: Utilization  
 averageUtilization: 50

## **7️⃣ Monitoring & Logging**

* Deploy **Prometheus + Grafana** from monitoring/
* Deploy **EFK Stack** from logging/

## **8️⃣ Network Policies**

network-policies/frontend-policy.yaml

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: allow-frontend-to-backend  
spec:  
 podSelector:  
 matchLabels:  
 app: backend  
 ingress:  
 - from:  
 - podSelector:  
 matchLabels:  
 app: frontend

## **✅ Execution Order**

1. kind create cluster
2. kubectl apply -f ingress/
3. kubectl apply -f cert-manager/
4. Deploy mongodb/
5. Deploy helm-charts for each service
6. Apply hpa-pdb/
7. Apply network-policies/
8. Setup monitoring and logging
9. Configure CI/CD
10. Test DR strategy

Would you like me to also include a **CI/CD Jenkins pipeline** for deploying these Helm charts automatically (build → push → deploy)?

========================================================================

Yes, you can run a **fully functional CI/CD Jenkins pipeline locally** for this Kubernetes project. Here’s how it would work:

## **1️⃣ Jenkins Setup (Local)**

* Run Jenkins in a **Docker container**:

docker run -d \  
 --name jenkins \  
 -p 8080:8080 -p 50000:50000 \  
 -v jenkins\_home:/var/jenkins\_home \  
 jenkins/jenkins:lts

* Install plugins:
  + **Kubernetes CLI**
  + **Pipeline**
  + **Docker Pipeline**
  + **GitHub**
  + **Helm**
  + **Blue Ocean (optional)**

## **2️⃣ Kubernetes Local Cluster**

* Since you’re using **KIND**, expose your Kubernetes config to Jenkins:

docker cp ~/.kube/config jenkins:/var/jenkins\_home/.kube/config

* Test connectivity:

docker exec -it jenkins kubectl get nodes

## **3️⃣ Jenkinsfile (CI/CD)**

Create ci-cd/Jenkinsfile:

pipeline {  
 agent any  
 environment {  
 KUBE\_CONFIG = '/var/jenkins\_home/.kube/config'  
 HELM\_HOME = '/usr/local/bin/helm'  
 REGISTRY = 'local-registry:5000'  
 }  
 stages {  
 stage('Checkout') {  
 steps {  
 git branch: 'main', url: '<https://github.com/youruser/k8s-ecommerce-project.git>'  
 }  
 }  
 stage('Build & Push Docker Image') {  
 steps {  
 sh '''  
 docker build -t $REGISTRY/product-service:${BUILD\_NUMBER} ./product-service  
 docker push $REGISTRY/product-service:${BUILD\_NUMBER}  
 '''  
 }  
 }  
 stage('Deploy with Helm') {  
 steps {  
 sh '''  
 helm upgrade --install product-service ./helm-charts/product-service \  
 --namespace ecommerce \  
 --set image.repository=$REGISTRY/product-service \  
 --set image.tag=${BUILD\_NUMBER}  
 '''  
 }  
 }  
 stage('Post-Deployment Check') {  
 steps {  
 sh 'kubectl rollout status deployment/product-service -n ecommerce'  
 }  
 }  
 }  
}

## **4️⃣ Local Docker Registry**

* To simulate ECR/GCR:

docker run -d -p 5000:5000 --name registry registry:2

* Tag and push images to this local registry.

## **5️⃣ Triggering the Pipeline**

* Add a GitHub webhook (or run manually).
* Each commit → build Docker image → push to local registry → deploy on KIND via Helm.

✅ **Result:**

* Works fully offline.
* Builds your microservices.
* Automatically upgrades Helm releases in your KIND cluster.

Would you like me to also include **GitHub Actions** (so later you can migrate the same pipeline to cloud CI/CD without changes)?

====================================================================

Yes, you can run a **fully functional CI/CD Jenkins pipeline locally** for this Kubernetes project. Here’s how it would work:

## **1️⃣ Jenkins Setup (Local)**

* Run Jenkins in a **Docker container**:

docker run -d \  
 --name jenkins \  
 -p 8080:8080 -p 50000:50000 \  
 -v jenkins\_home:/var/jenkins\_home \  
 jenkins/jenkins:lts

* Install plugins:
  + **Kubernetes CLI**
  + **Pipeline**
  + **Docker Pipeline**
  + **GitHub**
  + **Helm**
  + **Blue Ocean (optional)**

## **2️⃣ Kubernetes Local Cluster**

* Since you’re using **KIND**, expose your Kubernetes config to Jenkins:

docker cp ~/.kube/config jenkins:/var/jenkins\_home/.kube/config

* Test connectivity:

docker exec -it jenkins kubectl get nodes

## **3️⃣ Jenkinsfile (CI/CD)**

Create ci-cd/Jenkinsfile:

pipeline {  
 agent any  
 environment {  
 KUBE\_CONFIG = '/var/jenkins\_home/.kube/config'  
 HELM\_HOME = '/usr/local/bin/helm'  
 REGISTRY = 'local-registry:5000'  
 }  
 stages {  
 stage('Checkout') {  
 steps {  
 git branch: 'main', url: '<https://github.com/youruser/k8s-ecommerce-project.git>'  
 }  
 }  
 stage('Build & Push Docker Image') {  
 steps {  
 sh '''  
 docker build -t $REGISTRY/product-service:${BUILD\_NUMBER} ./product-service  
 docker push $REGISTRY/product-service:${BUILD\_NUMBER}  
 '''  
 }  
 }  
 stage('Deploy with Helm') {  
 steps {  
 sh '''  
 helm upgrade --install product-service ./helm-charts/product-service \  
 --namespace ecommerce \  
 --set image.repository=$REGISTRY/product-service \  
 --set image.tag=${BUILD\_NUMBER}  
 '''  
 }  
 }  
 stage('Post-Deployment Check') {  
 steps {  
 sh 'kubectl rollout status deployment/product-service -n ecommerce'  
 }  
 }  
 }  
}

## **4️⃣ Local Docker Registry**

* To simulate ECR/GCR:

docker run -d -p 5000:5000 --name registry registry:2

* Tag and push images to this local registry.

## **5️⃣ Triggering the Pipeline**

* Add a GitHub webhook (or run manually).
* Each commit → build Docker image → push to local registry → deploy on KIND via Helm.

✅ **Result:**

* Works fully offline.
* Builds your microservices.
* Automatically upgrades Helm releases in your KIND cluster.

Would you like me to also include **GitHub Actions** (so later you can migrate the same pipeline to cloud CI/CD without changes)?

======================================================================

Perfect—we’ll make it **100% beginner-friendly**. I’ll provide:

* **Full folder structure**
* **Every YAML, Helm, Jenkinsfile content**
* **Step-by-step commands**
* **What to check after each step**

## **📂 Folder Structure**

k8s-ecommerce-project/  
│── kind-cluster.yaml  
│── ingress/  
│ ├── ingress-controller.yaml  
│ ├── ingress-rules.yaml  
│── cert-manager/  
│ ├── cert-manager.yaml  
│ ├── cluster-issuer.yaml  
│── mongodb/  
│ ├── pv-pvc.yaml  
│ ├── statefulset.yaml  
│ ├── service.yaml  
│── helm-charts/  
│ ├── auth-service/  
│ │ ├── Chart.yaml  
│ │ ├── values.yaml  
│ │ ├── templates/deployment.yaml  
│ │ ├── templates/service.yaml  
│ ├── product-service/ (same as auth)  
│ ├── order-service/ (same as auth)  
│ ├── frontend/  
│ ├── Chart.yaml  
│ ├── values.yaml  
│ ├── templates/deployment.yaml  
│ ├── templates/service.yaml  
│── hpa-pdb/  
│ ├── hpa.yaml  
│ ├── pdb.yaml  
│── network-policies/  
│ ├── frontend-policy.yaml  
│ ├── backend-policy.yaml  
│── monitoring/  
│ ├── prometheus.yaml  
│ ├── grafana.yaml  
│── logging/  
│ ├── elasticsearch.yaml  
│ ├── fluentd.yaml  
│ ├── kibana.yaml  
│── ci-cd/  
│ ├── Jenkinsfile  
│ ├── github-actions.yaml  
│── disaster-recovery/  
│ ├── etcd-backup.md  
│ ├── mongo-backup.md  
│── README.md

## **1️⃣ kind-cluster.yaml**

kind: Cluster  
apiVersion: kind.x-k8s.io/v1alpha4  
nodes:  
- role: control-plane  
- role: worker  
- role: worker

**Command:**

kind create cluster --name ecommerce-cluster --config=kind-cluster.yaml  
kubectl get nodes

## **2️⃣ Ingress**

### **ingress/ingress-controller.yaml**

# NGINX Ingress Controller  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: ingress-nginx  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: ingress-nginx-controller  
 namespace: ingress-nginx  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app.kubernetes.io/name: ingress-nginx  
 template:  
 metadata:  
 labels:  
 app.kubernetes.io/name: ingress-nginx  
 spec:  
 containers:  
 - name: controller  
 image: k8s.gcr.io/ingress-nginx/controller:v1.9.0  
 args:  
 - /nginx-ingress-controller

### **ingress/ingress-rules.yaml**

apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
 name: ecommerce-ingress  
 annotations:  
 nginx.ingress.kubernetes.io/rewrite-target: /  
spec:  
 rules:  
 - host: ecommerce.local  
 http:  
 paths:  
 - path: /  
 pathType: Prefix  
 backend:  
 service:  
 name: frontend  
 port:  
 number: 80

## **3️⃣ Cert-Manager**

### **cert-manager/cert-manager.yaml**

kubectl apply -f <https://github.com/jetstack/cert-manager/releases/download/v1.12.0/cert-manager.yaml>

### **cert-manager/cluster-issuer.yaml**

apiVersion: cert-manager.io/v1  
kind: ClusterIssuer  
metadata:  
 name: letsencrypt-prod  
spec:  
 acme:  
 server: <https://acme-v02.api.letsencrypt.org/directory> email: [admin@example.com](mailto:admin@example.com) privateKeySecretRef:  
 name: letsencrypt-prod  
 solvers:  
 - http01:  
 ingress:  
 class: nginx

## **4️⃣ MongoDB**

### **mongodb/pv-pvc.yaml**

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: mongodb-pvc  
spec:  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 5Gi

### **mongodb/statefulset.yaml**

apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
 name: mongodb  
spec:  
 serviceName: mongodb  
 replicas: 3  
 selector:  
 matchLabels:  
 app: mongodb  
 template:  
 metadata:  
 labels:  
 app: mongodb  
 spec:  
 containers:  
 - name: mongo  
 image: mongo:6  
 ports:  
 - containerPort: 27017  
 volumeMounts:  
 - name: mongo-storage  
 mountPath: /data/db  
 volumeClaimTemplates:  
 - metadata:  
 name: mongo-storage  
 spec:  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 5Gi

### **mongodb/service.yaml**

apiVersion: v1  
kind: Service  
metadata:  
 name: mongodb  
spec:  
 ports:  
 - port: 27017  
 selector:  
 app: mongodb  
 clusterIP: None

## **5️⃣ Helm Charts (Example: auth-service)**

### **helm-charts/auth-service/Chart.yaml**

apiVersion: v2  
name: auth-service  
version: 0.1.0

### **helm-charts/auth-service/values.yaml**

replicaCount: 2  
image:  
 repository: auth-service  
 tag: latest  
resources:  
 requests:  
 cpu: 100m  
 memory: 256Mi  
 limits:  
 cpu: 500m  
 memory: 512Mi

### **helm-charts/auth-service/templates/deployment.yaml**

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: auth-service  
spec:  
 replicas: {{ .Values.replicaCount }}  
 selector:  
 matchLabels:  
 app: auth-service  
 template:  
 metadata:  
 labels:  
 app: auth-service  
 spec:  
 containers:  
 - name: auth  
 image: {{ .Values.image.repository }}:{{ .Values.image.tag }}  
 ports:  
 - containerPort: 8080

### **helm-charts/auth-service/templates/service.yaml**

apiVersion: v1  
kind: Service  
metadata:  
 name: auth-service  
spec:  
 ports:  
 - port: 80  
 targetPort: 8080  
 selector:  
 app: auth-service

(Repeat similar structure for product-service and order-service.)

### **✅ Next Steps**

* I can continue writing **all remaining files (frontend, monitoring, logging, HPA, PDB, Network Policies, CI/CD Jenkinsfile, GitHub Actions, Disaster Recovery)** in the same detailed manner.
* After that, we’ll create the **LinkedIn post draft**.

Do you want me to **continue generating all remaining files** now (it will be long but complete) or break it down phase-wise (e.g., Database first → Backend → Frontend)?

sudo snap install helm