Kubernetes Project Full Tutorial

This tutorial will guide you through the **Profile Project on Kubernetes** step by step, covering source code setup, manifests (Secrets, PVC, Deployments, Services, Ingress), cluster setup with Kops, GitHub repo integration, and complete configuration. Nothing is skipped—so you can do hands-on directly from this document.

---

1. Source Code Overview & Setup

Steps:

1. Open the repo:
2. Go to: `github.com/coder/profile-project`
3. Select **branch** `cube-app`.
4. Another branch will be used later (empty skeleton manifests).
5. What's inside `cube-app` branch:
6. Docker files, docker-compose file.
7. `cube-devs/` folder → contains all Kubernetes manifests.
8. Clone the repo locally:

```
# In VS Code
Click on Source Control → Clone Repository
Paste HTTPS URL from repo → Choose a folder (e.g., F:/CubeApp)
```

1. Switch to branch `cube-app`:
2. Contains `cube-devs` → all manifests.
3. Files include: Secret, PVC, Deployments (App, DB, RabbitMQ, Memcache), Services, Ingress.
4. Install **Kubernetes VS Code extension**:
5. Go to Extensions → search "Kubernetes" → Install.
6. Ignore `kubectl` error for now.

7. Architecture:

8. Secret → DB & RabbitMQ passwords.

9. PVC → Persistent storage.
10. Deployments → Tomcat app, MySQL DB, RabbitMQ, Memcache.
11. Services → ClusterIP (internal comm).
12. Ingress → external access.

---

2. Kubernetes Secret

We need secrets for DB & RabbitMQ passwords.

Steps:

1. In `cube-devs/secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
type: Opaque
data:
  db-pass: <ENCODED_DB_PASS>
  rmq-pass: <ENCODED_RMQ_PASS>
```

1. Encode passwords using Base64:

```
echo -n "wiproPass" | base64   # Example DB password
echo -n "guest" | base64       # RabbitMQ password
```

1. Use these encoded values inside `secret.yaml`.

---

3. Persistent Volume Claim (PVC)

DB pod needs storage → We'll use AWS EBS via default storage class.

Steps:

1. In `cube-devs/db-pvc.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pv-claim
  labels:
    app: wipro-db
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: default
  resources:
    requests:
      storage: 3Gi
```

---

4. MySQL Deployment & Service

We deploy DB using a custom image + PVC + Secret.

Deployment → `cube-devs/db-deploy.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wipro-db
  labels:
    app: wipro-db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wipro-db
  template:
    metadata:
      labels:
        app: wipro-db
    spec:
      containers:
        - name: wipro-db
          image: wiprocontainer/vprofiledb
          ports:
            - name: wipro-db-port
              containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
```

```
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: db-pass
          volumeMounts:
            - name: db-data
              mountPath: /var/lib/mysql
      volumes:
        - name: db-data
          persistentVolumeClaim:
            claimName: db-pv-claim
      initContainers:
        - name: init-clean
          image: busybox:latest
          command: ["rm", "-rf", "/var/lib/mysql/lost+found"]
          volumeMounts:
            - name: db-data
              mountPath: /var/lib/mysql
```

Service → `cube-devs/db-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: wipro-db
spec:
  type: ClusterIP
  selector:
    app: wipro-db
  ports:
    - port: 3306
      targetPort: wipro-db-port
```

---

5. Memcache Deployment + Service

Deployment → `cube-devs/mc-deploy.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wipro-mc
  labels:
    app: wipro-mc
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wipro-mc
  template:
    metadata:
      labels:
        app: wipro-mc
    spec:
      containers:
        - name: wipro-mc
          image: memcached
          ports:
            - name: wipro-mc-port
              containerPort: 11211
```

Service → `cube-devs/mc-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: wipro-cache01
spec:
  type: ClusterIP
  selector:
    app: wipro-mc
  ports:
    - port: 11211
      targetPort: wipro-mc-port
```

---

6. RabbitMQ Deployment + Service

Deployment → `cube-devs/rmq-deploy.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wipro-mq
  labels:
    app: wipro-mq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wipro-mq
  template:
    metadata:
      labels:
        app: wipro-mq
    spec:
      containers:
        - name: wipro-mq
          image: rabbitmq
          ports:
            - name: wipro-rmq-port
              containerPort: 5672
          env:
            - name: RABBITMQ_DEFAULT_USER
              value: guest
            - name: RABBITMQ_DEFAULT_PASS
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: rmq-pass
```

Service → `cube-devs/rmq-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: wipro-mq
spec:
  type: ClusterIP
  selector:
    app: wipro-mq
  ports:
    - port: 5672
      targetPort: wipro-rmq-port
```

---

7. Tomcat Application Deployment + Service

Deployment → `cube-devs/app-deploy.yaml`

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

```
metadata:
  name: vpro-app
  labels:
    app: vpro-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vpro-app
  template:
    metadata:
      labels:
        app: vpro-app
    spec:
      containers:
        - name: vpro-app
          image: wiprocontainer/vprofileapp
          ports:
            - name: app-port
              containerPort: 8080
```

Service → `cube-devs/app-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  type: ClusterIP
  selector:
    app: vpro-app
  ports:
    - port: 8080
      targetPort: app-port
```

---

8. Ingress Controller & Rules

We need external access for Tomcat app → use **Ingress**.

Ingress Controller Setup

- Install **nginx ingress controller** using official YAML.
- It manages an external Load Balancer (e.g., AWS ALB) and connects to ClusterIP services.

Ingress Rule → `cube-devs/app-ingress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vpro-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
    - host: vprofile.my-domain.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: webapp-service
                port:
                  number: 8080
```

- Update host with your **domain** (GoDaddy → Route53 → LoadBalancer DNS).
- Supports **path-based routing**: `/login`, `/videos`, `/payments` etc.
- One ALB can handle multiple services via rules.

---

9. Cluster Setup with Kops

1. SSH into your **Kops EC2 instance**.
2. Run:

```
kops create cluster --name <cluster-name> --state s3://<bucket-name> --
zones=ap-south-1a --node-count=2 --node-size=t2.micro --master-
size=t2.micro --dns-zone <your-domain>

kops update cluster --name <cluster-name> --state s3://<bucket-name> --yes
--admin
```

3. Wait for cluster creation.

Prerequisites:

- Hosted zone in Route53.
- Records updated in GoDaddy to point to Route53.

- S3 bucket for kops state store.
- AWS access keys configured.

---

10. Push Code to GitHub

1. Create repo in GitHub (public).
2. Upload only `cube-devs/` folder and optionally Dockerfiles.
3. Commit changes.
4. Clone inside Kops instance:

   ```
   git clone https://github.com/<your-repo>.git
   cd <repo>
   ```

---

11. Apply All Manifests

```
kubectl apply -f cube-devs/
```

Verify:

```
kubectl get all
kubectl describe ingress vpro-ingress
```

✅With this, you've:

- Set up **Secrets, PVC, DB, Memcache, RabbitMQ, Tomcat app**.
- Configured **Ingress Controller** for external access.
- Learned **path/host-based routing** in Kubernetes.
- Pushed definitions to GitHub and deployed via Kops cluster.

⛎Your project is now fully running with internal + external access!