1. Run SonarQube locally (Docker):

```
docker run -d --name sonarqube -p 9000:9000 sonarqube:9.9-community
sonar-scanner -Dsonar.projectKey=metro -Dsonar.sources=src
```

sonar-project.properties

```
# sonar-project.properties
sonar.projectKey=metro
sonar.projectName=MetroRide
sonar.projectVersion=1.0
sonar.sources=src
sonar.java.binaries=target/classes
sonar.sourceEncoding=UTF-8
# Optional: coverage reports if you have them
# sonar.java.coveragePlugin=jacoco
# sonar.junit.reportPaths=target/surefire-reports
```

---

2. — Kubernetes Deployment (Task 2)

`deployment.yaml` (3 replicas)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metroride-deployment
  labels:
    app: metroride
spec:
  replicas: 3
  selector:
    matchLabels:
      app: metroride
  template:
    metadata:
      labels:
        app: metroride
    spec:
      containers:
        - name: metroride
          image: yourrepo/metroride:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
          resources:
            requests:
              cpu: "100m"
              memory: "128Mi"
            limits:
              cpu: "500m"
              memory: "512Mi"
```

`service-nodeport.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: metroride-service
spec:
  type: NodePort
  selector:
    app: metroride
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
      nodePort: 30080
```

Apply in Minikube

```
kubectl apply -f deployment.yaml
kubectl apply -f service-nodeport.yaml
```

Commands to capture (provide screenshot/printout)

```
kubectl get pods -o wide
kubectl get svc metroride-service
```

Example expected `kubectl get pods` output (sample)

```
NAME                                   READY   STATUS    RESTARTS   AGE
metroride-deployment-5d9d4d7f6d-abc12  1/1     Running   0          2m
metroride-deployment-5d9d4d7f6d-def34  1/1     Running   0          2m
metroride-deployment-5d9d4d7f6d-ghi56  1/1     Running   0          2m
```

---

3 — HPA (Autoscaling based on CPU) (Task 3, 7 marks)

Pre-reqs

- Metrics server installed (`minikube addons enable metrics-server`) or install `metrics-server` in cluster.

`hpa.yaml`

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

Apply & inspect

```
kubectl apply -f hpa.yaml
kubectl get hpa metroride-hpa
kubectl describe hpa metroride-hpa
```

What to provide

- `hpa.yaml` file.
- **Screenshot** of `kubectl get hpa` or `kubectl describe hpa` showing targets & current utilization.

---

4 — KEDA Event-based Scaling (Task 4, 8 marks)

KEDA scales based on external events (queues, Kafka, etc.). If you don't have a real queue available, a mock ScaledObject is acceptable for the assignment.

Install KEDA (example)

```
kubectl apply -f
https://github.com/kedacore/keda/releases/download/v2.10.0/keda-2.10.0.yaml
# or use helm install (if helm available)
```

Example ScaledObject (mock) — scales deployment based on a queue length (RabbitMQ example)

`scaledobject-rabbitmq.yaml`

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: metroride-rmq-scaledobject
  labels:
    app: metroride
spec:
  scaleTargetRef:
    name: metroride-deployment
  minReplicaCount: 1
  maxReplicaCount: 10
  cooldownPeriod: 30
  pollingInterval: 15
  triggers:
  - type: rabbitmq
    metadata:
      queueName: metroride-queue
      host: RabbitMQConnectionStringPlaceholder
      queueLength: "5"
```

- Replace `host` with `amqp://user:pass@rabbitmq-host:5672/` if you have RabbitMQ.
- For Azure Queue, Redis, Kafka etc., KEDA supports many triggers — pick one taught in class.

Apply & inspect

```
kubectl apply -f scaledobject-rabbitmq.yaml
kubectl get scaledobject
kubectl describe scaledobject metroride-rmq-scaledobject
```

What to provide

- `scaledobject-rabbitmq.yaml`
- **Screenshot** of `kubectl get scaledobject` / KEDA logs showing scaled activity (or the ScaledObject created).

Rolling Update (in `deployment-rolling.yaml`)

Rolling is the default strategy — incremental replacement.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metroride-rolling
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
      app: metroride-rolling
  template:
    metadata:
      labels:
        app: metroride-rolling
    spec:
      containers:
      - name: metroride
        image: yourrepo/metroride:v2
        ports:
        - containerPort: 8080
```

`deployment-blue.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metroride-blue
  labels:
    app: metroride
    version: blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: metroride
      version: blue
  template:
    metadata:
      labels:
        app: metroride
        version: blue
    spec:
      containers:
      - name: metroride
        image: yourrepo/metroride:v1
        ports:
        - containerPort: 8080
```

`deployment-green.yaml` (new version)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metroride-green
  labels:
    app: metroride
    version: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: metroride
      version: green
  template:
    metadata:
      labels:
        app: metroride
        version: green
    spec:
      containers:
      - name: metroride
        image: yourrepo/metroride:v2
        ports:
        - containerPort: 8080
```

`service-bluegreen.yaml` (service points to the chosen version via label selector)

```
apiVersion: v1
kind: Service
metadata:
  name: metroride-service
spec:
  type: NodePort
  selector:
    app: metroride
    version: blue   # switch to 'green' when you want to cutover
  ports:
  - port: 8080
    targetPort: 8080
    nodePort: 30080
```

Explanation (brief): Blue/Green uses two separate sets of pods (blue = current, green = new). Deploy green, run tests/health checks, then switch the Service selector from `version: blue` to `version: green` to cutover instantly; rollback is a simple selector change.

---

6 — Cloud Essentials (Task 6, 10 marks) — choose ONE

I'll give steps for the common options. Capture screenshots of console / created resources.

Option A: Create a VM and deploy JAR (example: AWS EC2)

1. Launch EC2 instance (Amazon Linux 2) via AWS Console.
2. SSH to instance:

```
ssh -i "key.pem" ec2-user@<ec2-public-ip>
```

3. Install Java and copy JAR:

```
sudo yum update -y
sudo yum install -y java-17-openjdk
# copy jar via scp
scp -i key.pem target/metroride.jar ec2-user@<ec2-public-ip>:/home/ec2-user/
# on server
java -jar metroride.jar
```

4. Open port 8080 in Security Group. Provide screenshot of EC2 instance list and running app endpoint `http://<ec2-public-ip>:8080/actuator/health`.

Option B: Create a Storage bucket and upload artifact (GCP example)

```
gsutil mb gs://metroride-artifacts-<your-id>
gsutil cp target/metroride.jar gs://metroride-artifacts-<your-id>/
```

Provide screenshot of the bucket and the uploaded artifact.

Option C: Create a VPC / VPN — minimal steps (AWS)

- Create VPC with a public subnet and route table in AWS Console — capture screenshot of VPC dashboard.

Provide one of the above with screenshots of consoles / commands as proof.

Checklist of deliverables to submit (exactly what exam rubric asks)

1. **SonarQube**
   - `sonar-project.properties` (provided above)
   - Screenshot of SonarQube dashboard (run locally and capture `http://localhost:9000`)
2. **Kubernetes Deployment**
   - `deployment.yaml` (provided above)
   - `service-nodeport.yaml` (provided above)
   - Output/screenshot: `kubectl get pods`, `kubectl get svc`
3. **HPA**
   - `hpa.yaml` (provided)
   - Screenshot of `kubectl get hpa` / `kubectl describe hpa`
4. **KEDA**
   - `scaledobject-rabbitmq.yaml` (provided) or one matching whichever trigger you implement
   - Screenshot of `kubectl get scaledobject` / KEDA controller logs
5. **Deployment Strategies**
   - `deployment-rolling.yaml` (provided)
   - `deployment-blue.yaml`, `deployment-green.yaml`, `service-bluegreen.yaml` (provided)
   - Short explanation (included)
6. **Cloud Task**
   - One of: VM steps + screenshot of instance + app URL, OR bucket screenshot with uploaded JAR, OR VPC screenshot

---

Extra tips to score full marks

- For Sonar: include metrics like **Bugs, Vulnerabilities, Code Smells, Coverage** in the screenshot.
- For HPA: show `kubectl top pods` during load to demonstrate scaling (you can use `stress` or a load generator like `hey`/`wrk`).
- For KEDA: if you don't have a real queue, simulate by creating the queue and pushing items; show scaling events in KEDA operator logs.
- For Blue/Green: show `kubectl get svc -o yaml` before and after switching selector, proving cutover.
- For Cloud: include timestamps in screenshots and the console resource ARN/IDs.

Quick commands cheat-sheet

Build image & load into Minikube:

```
# build
docker build -t yourrepo/metroride:latest .
# either push to registry or
minikube image load yourrepo/metroride:latest
```

Apply all K8s manifests:

```
kubectl apply -f deployment.yaml
kubectl apply -f service-nodeport.yaml
kubectl apply -f hpa.yaml
kubectl apply -f scaledobject-rabbitmq.yaml   # after installing KEDA
```

Obtain outputs (capture these as screenshots or copy-paste into a file):

```
kubectl get pods -o wide
kubectl get svc
kubectl get hpa
kubectl get scaledobject
kubectl describe hpa metroride-hpa
kubectl describe scaledobject metroride-rmq-scaledobject
```
