🔪 Lab 6.2 – Working with CRDs & Kro Reconciliation

🌟 Objectives

- Discover and explore CustomResourceDefinitions (CRDs)
- Deploy a simple CRD and Custom Resource
- Install and use **Kro** to demonstrate reconciliation via `ResourceGraphDefinition`

🔧 Prerequisites

- A running **Kind** cluster
- `kubectl` access
- Kro installed in the cluster

📅 Part 1: Basic CRD Creation and Usage

✏️ Step 1 – Discover CRDs

```
kubectl get crds
kubectl explain <crd-name>
```

✏️ Step 2 – Create a CRD for `Widget`

Create the file `widget-crd.yaml`:

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: widgets.training.io
spec:
  group: training.io
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                size:
                  type: string
  scope: Namespaced
  names:
    plural: widgets
    singular: widget
    kind: Widget
```

Apply it:

```
kubectl apply -f module-6/manifests/widget-crd.yaml
```

➕ Step 3 – Create a Custom Resource

Create the file `widget.yaml`:

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```yaml
apiVersion: training.io/v1
kind: Widget
metadata:
  name: my-widget
spec:
  size: large
```

Apply it:

```
kubectl apply -f module-6/manifests/widget.yaml
kubectl get widget
```

📅 Part 2: Reconciliation with Kro

✅ Step 1 – Install Kro

```
export KRO_VERSION=$(curl -sL \
    https://api.github.com/repos/kro-run/kro/releases/latest
| \
    jq -r '.tag_name | ltrimstr("v")'
  )

helm install kro oci://ghcr.io/kro-run/kro/kro \
  --namespace kro \
  --create-namespace \
  --version=${KRO_VERSION}
```

Wait for the controller pod to be ready:

```
kubectl get pods -n kro
```

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

🔧 Step 2 – Define a `ResourceGraphDefinition`

Create the file `rgd-app.yaml`:

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```yaml
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: my-color-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: Application
    spec:
      name: string
      image: string | default="pj3677/color-app:latest"
      color: string | default="red"
      commonlabel: string | default="color-application"
    status:
      deploymentConditions: ${deployment.status.conditions}
      availableReplicas: ${deployment.status.availableReplic
as}
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: ${schema.spec.name}
          labels:
            purpose: ${schema.spec.commonlabel}
        spec:
          replicas: 2
          selector:
            matchLabels:
              app: ${schema.spec.name}
          template:
            metadata:
              labels:
                app: ${schema.spec.name}
                purpose: ${schema.spec.commonlabel}
            spec:
              containers:
                - name: ${schema.spec.name}
                  image: ${schema.spec.image}
```

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```
                    ports:
                      - containerPort: 3000
                    env:
                      - name: BG_COLOR
                        value: ${schema.spec.color}
    - id: service
      template:
        apiVersion: v1
        kind: Service
        metadata:
          name: ${schema.spec.name}-service
          labels:
            purpose: ${schema.spec.commonlabel}
        spec:
          selector: ${deployment.spec.selector.matchLabels}
          ports:
            - protocol: TCP
              port: 3000
              targetPort: 3000
```

Apply it:

```
kubectl apply -f module-6/manifests/rgd-app.yaml
kubectl get resourcegraphdefinition
kubectl get crds
```

📅 Step 3 – Instantiate the `App` Custom Resource

Create the file `color-lightgreen-app.yaml`:

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```yaml
apiVersion: kro.run/v1alpha1
kind: Application
metadata:
  name: color-lightgreen
spec:
  name: color-lightgreen-app
  color: "lightgreen"
```

Create the file `color-red-app.yaml`:

```yaml
apiVersion: kro.run/v1alpha1
kind: Application
metadata:
  name: color-red
spec:
  name: color-red-app
  color: "red"
  commonLabel: "red-application"
```

Apply it:

```
kubectl apply -f module-6/manifests/color-lightgreen-app.yam
l
kubectl apply -f module-6/manifests/color-red-app.yaml
```

🔎 Step 4 – Validate

Check the generated resources:

```
kubectl get deploy
kubectl get svc
kubectl get pods
```

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

Access the app via the NodePort using the IP of any Kubernetes node.

🧠 Challenge Task

Create a new `App` instance:

- Use a different name (e.g. `color-alt`)
- Observe how Kro reconciles the Deployment + Service automatically

---

Cleanup

```
kubectl delete -f module-6/manifests/widget.yaml
kubectl delete -f module-6/manifests/rgd-app.yaml
kubectl delete -f module-6/manifests/color-lightgreen-app.ya
ml
kubectl delete -f module-6/manifests/color-red-app.yaml
helm uninstall kro -n kro
kubectl delete namespace kro
kubectl delete -f module-6/manifests/widget-crd.yaml
```

🎉 **End of Lab 6.2** – You explored CRDs and implemented dynamic resource reconciliation with Kro!

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30