Deploy the application

We will install the Nginx ingress controller by following the steps outlined in the official documentation: https://kind.sigs.k8s.io/docs/user/ingress/

We can do the following:

```
# Delete the existing cluster (if any)
kind delete cluster
# Create a new cluster the follows a specific configuration
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF
# Wait untilt the cluster is up and running and install the nginx
ingress controller
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/main/deploy/static/provider/kind/deploy.yaml
```

Wait for the ingress controller to be ready:

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

Create the application

We need a microservices application that contains three services:

- Admin -> www.example.com/admin
- API -> api.example.com
- Main -> www.example.com

Let's create the pods for the three microservices:

The www service

```yaml
# www.yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    app: www
spec:
  containers:
  - name: www
    image: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: www
spec:
  selector:
    app: www
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

The API pod

```yaml
---
# api-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: api
  labels:
    app: api
spec:
  containers:
  - name: api
    image: ealen/echo-server:latest
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: api
spec:
  selector:
    app: api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

The admin pod

```yaml
---
# admin.yaml
apiVersion: v1
kind: Pod
metadata:
  name: admin
  labels:
    app: admin
spec:
  containers:
  - name: admin
    image: ealen/echo-server:latest
    env:
    - name: ECHO_SERVER_BASE_PATH
      value: "/admin"
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: admin
spec:
  selector:
    app: admin
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

The ingress object

```yaml
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
spec:
  ingressClassName: nginx
  rules:
  - host: www.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: www
            port:
              number: 80
      - path: /admin
        pathType: Exact
        backend:
          service:
            name: admin
            port:
              number: 80
  - host: api.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api
            port:
              number: 80
```

Implementation

Apply all the manifests to the cluster.

Check the resources that we have:

```
$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
admin   1/1     Running   0          5m22s
api     1/1     Running   0          5m22s
www     1/1     Running   0          4m50s
$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
admin        ClusterIP   10.96.138.203   <none>        80/TCP
5m33s
api          ClusterIP   10.96.129.175   <none>        80/TCP
5m33s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   138m
www          ClusterIP   10.96.31.131    <none>        80/TCP
5m33s
$ kubectl get ing
NAME      CLASS   HOSTS                           ADDRESS     PORTS
  AGE
ingress   nginx   www.example.com,api.example.com   localhost   80
   98m
```

Add the following entries to `/etc/hosts`

```
127.0.0.1 api.example.com
127.0.0.1 www.example.com
```

Testing

Test the application by going to www.example.com, www.example.com/admin, and api.example.com

Notice how www.example.com/admin ONLY is routed but not www.example.com/admin/something because of the `Exact` path type.

However, if we type api.example.com/something, it will get routed because the path type is `Prefix`.