TABLE OF CONTENTS

USING ORACLE CLOUD CONTAINER ENGINE FOR KUBERNETES (OKE) AND THE MICRONAUT FRAMEWORK

Learn how to deploy your Micronaut applications on the Oracle Cloud Container Engine for Kubernetes (OKE).

Authors: Nemanja Mikic

Micronaut Version: 3.7.3

1. Getting Started

In this guide, we will deploy three microservices on the **Oracle Cloud Container Engine for Kubernetes (OKE)**. We will use Kubernetes Service discovery and Distributed configuration to wire up our microservices.

You will discover how the Micronaut framework eases Kubernetes integration and deployment to OKE.

2. What you will need

To complete this guide, you will need the following:

- Some time on your hands
- A decent text editor or IDE
- JDK 17 or greater installed with `JAVA_HOME` configured appropriately
- **Docker**.
- A paid or free trial Oracle Cloud account (create an account at **signup.oraclecloud.com**)

  Your Oracle Cloud account must be a paid account or trial with credits available because there isn't currently a free-tier option for OKE.

- We'll use the OCI command line to authenticate ourselves. If you don't have it already, install the **Oracle Cloud CLI** and run `oci setup config`.

Some of the following commands use `jq`

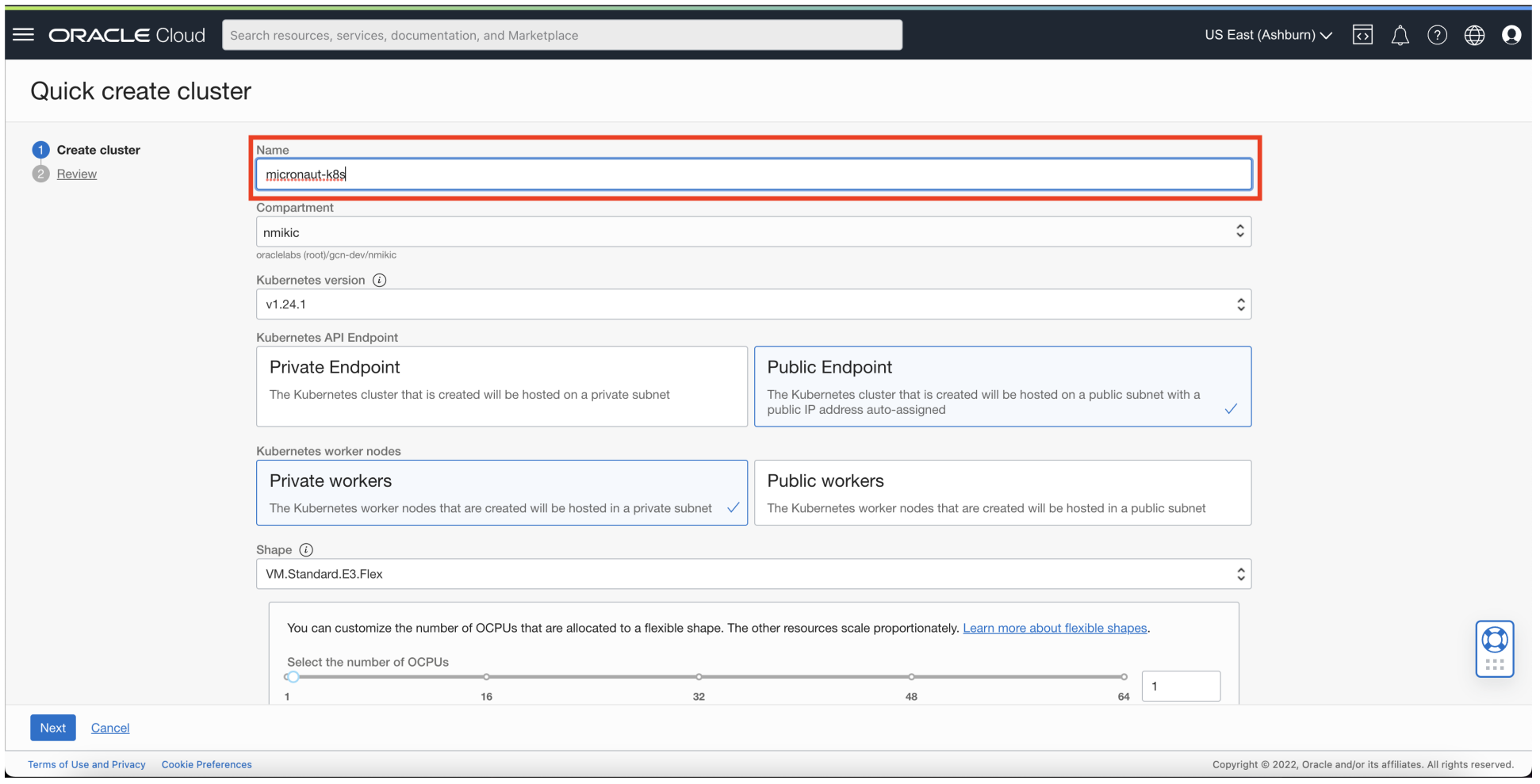
> jq is a lightweight and flexible command-line JSON processor

3. The Application

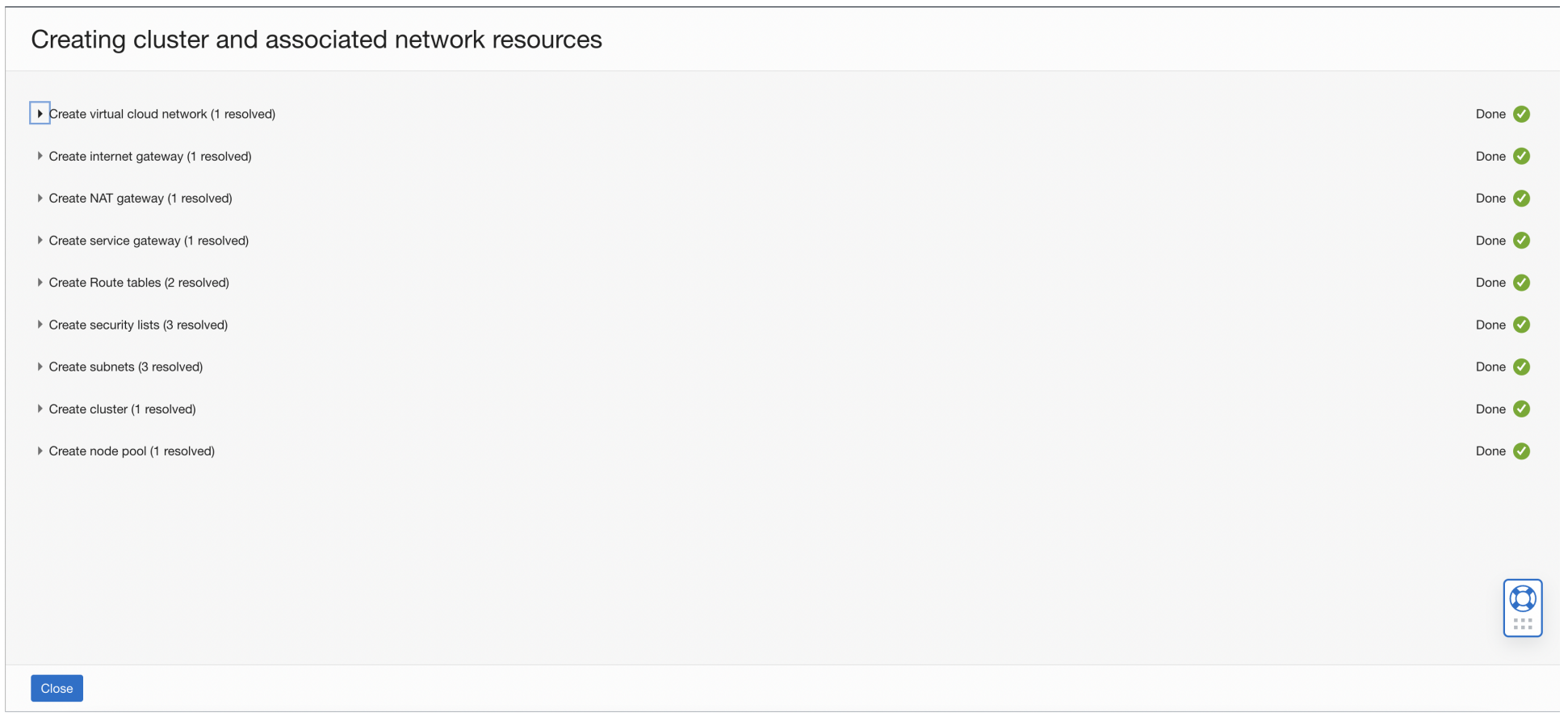
Download the complete solution of the **Kubernetes and the Micronaut Framework** guide. You will use same microservices (users, orders and api) as a starting point.

4. Create Oracle Cloud Kubernetes Cluster

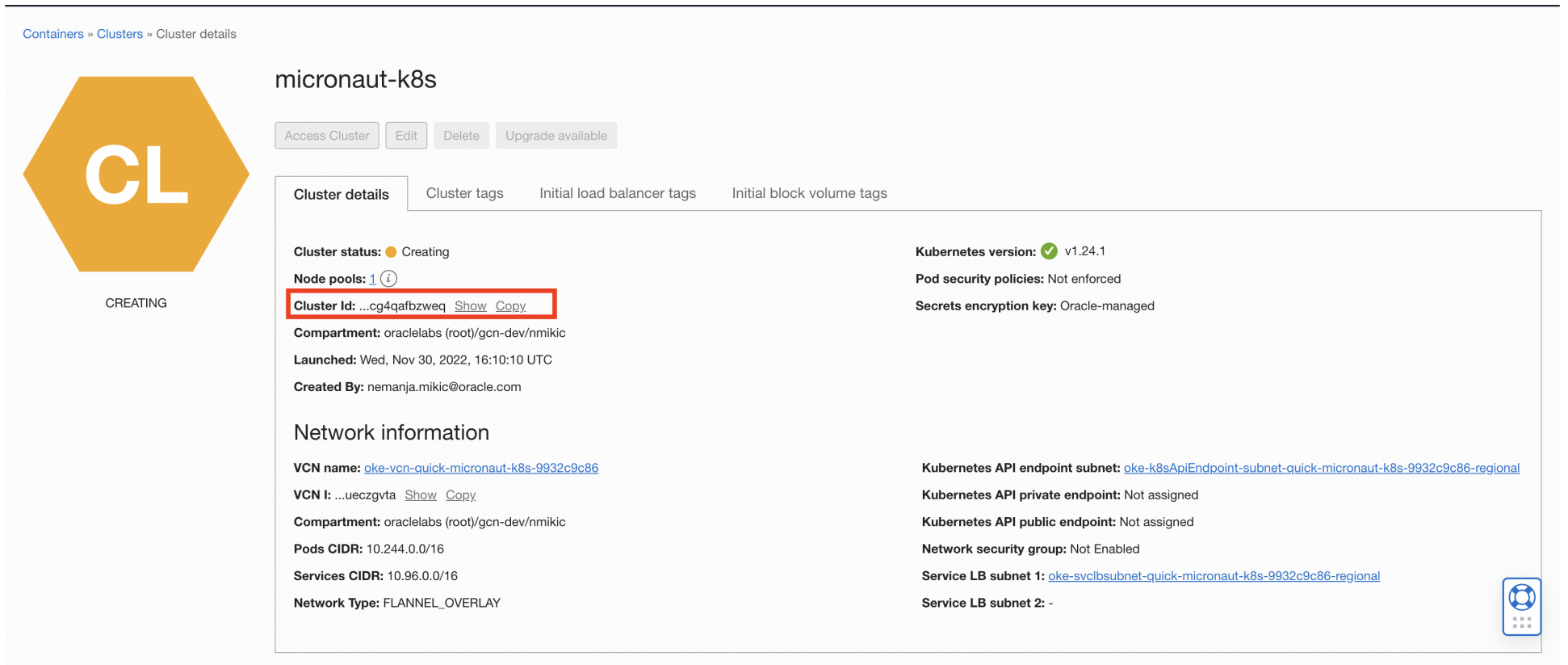
We will use Quick create cluster option on OKE to create Kubernetes Cluster. To start browse **Quick Kubernetes Clusters (OKE)**.



Choose name for kubernetes cluster, we chose to name it `micronaut-k8s`.



Wait to finish all resources to be created.



Copy `Cluster Id` you will need it later.

5. Prepare and deploy microservices

We will define some variables to make deploying process easier. In `OCI_USER_ID` store your user ocid, you can find it in your oci config file. In `OCI_TENANCY_NAME` store your tenancy name. In `OCI_USERNAME` we will store username in format <tenancy>/<username>. We can reuse `OCI_TENANCY_NAME` and only edit <username> part. Store your region key code in `OCI_REGION` for an example us-phoenix-1. In `OCI_CLUSTER_ID` put the Cluster Id that you copied before.

```
export OCI_USER_ID="ocid1.user.oc1..aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
export OCI_TENANCY_NAME="<tenancy-id>"
export OCI_USERNAME="$OCI_TENANCY_NAME/<username>"
export OCI_REGION="<region-key>"
export OCI_CLUSTER_ID="ocid1.cluster.oc1.iad.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

Copy

We have to create `AUTH_TOKEN` to be able to authenticate to Oracle Cloud Container Registry.

```
export AUTH_TOKEN=$(oci iam auth-token create --user-id $OCI_USER_ID --description k8s-micronaut | jq -r '.data.t
```

Copy

Run next command to log in to `ocir.io` (Oracle Cloud Container Registry):

```
docker login $OCI_REGION.ocir.io -u $OCI_USERNAME -p $AUTH_TOKEN
```

Copy

5.1. Users microservice

Edit `k8s.yml` inside `users` service.

/users/k8s.yml

Copy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: "users"
spec:
  selector:
    matchLabels:
      app: "users"
  template:
    metadata:
      labels:
        app: "users"
    spec:
      serviceAccountName: micronaut-service
      containers:
        - name: "users"
          image: <region-key>.ocir.io/<tenancy-name>/users:last (1)
          imagePullPolicy: Always (2)
          ports:
            - name: http
              containerPort: 8080
          readinessProbe:
            httpGet:
              path: /health/readiness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
          livenessProbe:
            httpGet:
              path: /health/liveness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
            failureThreshold: 10
      imagePullSecrets:
        - name: ocirsecret (3)
---
apiVersion: v1
kind: Service
metadata:
  name: "users"
spec:
  selector:
    app: "users"
  type: NodePort
  ports:
    - protocol: "TCP"
      port: 8080
```

(1) Image name that exists in OCI docker registry. Change the <region-key> to your region and change the <tenancy-name> to your tenancy name.

(2) Change imagePullPolicy to Always

(3) Secret needed to pull images from Oracle Docker registry

Tag an existing users microservice image.

```
docker tag users:latest $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/users:latest
```

Copy

Push tagged users microservice image to remote repository.

```
docker push $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/users:latest
```

Copy

5.2. Orders microservice

Edit `k8s.yml` inside `orders` service.

/orders/k8s.yml

Copy

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: "orders"
spec:
  selector:
    matchLabels:
      app: "orders"
  template:
    metadata:
      labels:
        app: "orders"
    spec:
      serviceAccountName: micronaut-service
      containers:
        - name: "orders"
          image: <region-key>.ocir.io/<tenancy-name>/orders:last (1)
          imagePullPolicy: Always (2)
          ports:
            - name: http
              containerPort: 8080
          readinessProbe:
            httpGet:
              path: /health/readiness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
          livenessProbe:
            httpGet:
              path: /health/liveness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
            failureThreshold: 10
      imagePullSecrets:
        - name: ocirsecret (3)
---
apiVersion: v1
kind: Service
metadata:
  name: "orders"
spec:
  selector:
    app: "orders"
  type: NodePort
  ports:
    - protocol: "TCP"
      port: 8080
```

(1) Image name that exists in OCI docker registry. Change the <region-key> to your region and change the <tenancy-name> to your tenancy name.

(2) Change imagePullPolicy to Always

(3) Secret needed to pull images from Oracle Docker registry

Tag an existing orders microservice image.

```
docker tag orders:latest $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/orders:latest
```

Copy

Push tagged orders microservice image to remote repository.

```
docker push $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/orders:latest
```

Copy

5.3. API microservice

Edit `k8s.yml` inside `api` service.

/api/k8s.yml

Copy

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: "api"
spec:
  selector:
    matchLabels:
      app: "api"
  template:
    metadata:
      labels:
        app: "api"
    spec:
      serviceAccountName: micronaut-service
      containers:
        - name: "api"
          image: <region-key>.ocir.io/<tenancy-name>/api:last (1)
          imagePullPolicy: Always (2)
          ports:
            - name: http
              containerPort: 8080
          readinessProbe:
            httpGet:
              path: /health/readiness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
          livenessProbe:
            httpGet:
              path: /health/liveness
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 3
            failureThreshold: 10
      imagePullSecrets:
        - name: ocirsecret (3)
---
apiVersion: v1
kind: Service
metadata:
  name: "api"
spec:
  selector:
    app: "api"
  type: LoadBalancer
  ports:
    - protocol: "TCP"
      port: 8080
```

(1) Image name that exists in OCI docker registry. Change the <region-key> to your region and change the <tenancy-name> to your tenancy name.

(2) Change imagePullPolicy to Always

(3) Secret needed to pull images from Oracle Docker registry

Tag an existing api microservice image.

```
docker tag api:latest $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/api:latest
```

Copy

Push tagged api microservice image to remote repository.

```
docker push $OCI_REGION.ocir.io/$OCI_TENANCY_NAME/api:latest
```

Copy

5.4. Deploy services to OKE

Create folder for `kubectl` configuration.

```
mkdir –p $HOME/.kube
```

Copy

Generate `kubectl` configuration for authentication to OKE.

```
oci ce cluster create-kubeconfig --cluster-id $OCI_CLUSTER_ID --file $HOME/.kube/config --region $OCI_REGION --token-version 2.0.0  --kube-endpoint PUBLIC_ENDPOINT
```

Set `KUBECONFIG` to created config file. This variable is consumed by `kubectl`.

```
export KUBECONFIG=$HOME/.kube/config
```

Create `ocirsecret` secret that will be used for authentication to Oracle Cloud Container Registry. This is needed because OKE needs docker credentials to be able to pull microservices images.

```
kubectl create secret docker-registry ocirsecret --docker-server=$OCI_REGION.ocir.io --docker-username=$OCI_USERNAME --docker-password=$AUTH_TOKEN
```

Deploy `auth.yml` file that we created in [Quick Kubernetes Clusters (OKE)] guide.

```
kubectl create -f auth.yml
```

Run next command to deploy users microservice:

```
kubectl create -f users/k8s.yml
```

Run next command to deploy orders microservice:

```
kubectl create -f orders/k8s.yml
```

Run next command to deploy api microservice:

```
kubectl create -f api/k8s.yml
```

6. Test integration between applications deployed on OKE

Run next command to check status of created pods and make sure that all have status "Running":

```
kubectl get pods
```

```
NAME                      READY   STATUS    RESTARTS   AGE
api-6fb4cd949f-kxxx8      1/1     Running   0          2d1h
orders-595887ddd6-6lzp4   1/1     Running   0          2d1h
users-df6f78cd7-lgnzx     1/1     Running   0          2d1h
```

Run next command to check status of created services:

```
kubectl get services
```

```
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)             AGE
api          LoadBalancer   10.96.70.48    129.159.92.209   8080:31690/TCP      2d1h
kubernetes   ClusterIP      10.96.0.1      <none>           443/TCP,12250/TCP   7d23h
orders       NodePort       10.96.94.130   <none>           8080:31245/TCP      2d1h
users        NodePort       10.96.34.174   <none>           8080:30790/TCP      2d1h
```

Run next command to get url of api service:

```
export API_URL=http://$(kubectl get svc api -o json | jq -r .status.loadBalancer.ingress[0].ip):8080
```

Run a cURL command to create user through api:

```
curl -X "POST" "$API_URL/api/users" -H 'Content-Type: application/json; charset=utf-8' -d '{ "first_name": "Nemanja",
"last_name": "Mikic", "username": "nmikic" }'
```

{"id":1,"username":"nmikic","first_name":"Nemanja","last_name":"Mikic"}

Copy

Run a cURL command to create order through api:

```
curl -X "POST" "$API_URL/api/orders" -H 'Content-Type: application/json; charset=utf-8' -d '{ "user_id": 1, "item_
[1,2] }'
```

Copy

{"id":1,"user":{"first_name":"Nemanja","last_name":"Mikic","id":1,"username":"nmikic"},"items":
[{"id":1,"name":"Banana","price":1.5},{"id":2,"name":"Kiwi","price":2.5}],"total":4.0}

Copy

Run a cURL to list created orders:

```
curl "$API_URL/api/orders" -H 'Content-Type: application/json; charset=utf-8'
```

Copy

[{"id":1,"user":{"first_name":"Nemanja","last_name":"Mikic","id":1,"username":"nmikic"},"items":
[{"id":1,"name":"Banana","price":1.5},{"id":2,"name":"Kiwi","price":2.5}],"total":4.0}]

Copy

We can try to place an order with user that doesn't exist (with id 100). Run a cURL command:

```
curl -X "POST" "$API_URL/api/orders" -H 'Content-Type: application/json; charset=utf-8' -d '{ "user_id": 100,
"item_ids": [1,2] }'
```

Copy

{"message":"Bad Request","_links":{"self":[{"href":"/api/orders","templated":false}]},"_embedded":{"errors":
[{"message":"User with id 100 doesn't exist"}]}}

Copy

7. Next steps

Explore more features with **Micronaut Guides**.

Read more about **Micronaut Kubernetes** module.

Read more about **Oracle Container Engine for Kubernetes (OKE)**