## Table of Contents



# Microservices Deployment Lab

**Goals**

- Access Red Hat® OpenShift® environment
- Deploy microservices applications to OpenShift Container Platform
- Observe automatic injection of Service Mesh sidecar into each microservice

# 1. Set-up

There are three microservices in this lab that you will deploy to OpenShift. In a later lab of this course, you will manage the interactions between these microservices using Red Hat OpenShift Service Mesh.

Here is the application architecture of the microservices:


1. Switch to *user1* OpenShift user:

```
oc login -u user1 -p r3dh4t1!
```

2. Create a tutorial project:

```
echo "export OCP_TUTORIAL_PROJECT=user1-tutorial" >> $HOME/.bashrc
source $HOME/.bashrc

oc new-project $OCP_TUTORIAL_PROJECT
```

3. Clone the course lab assets to the remote *bastion* node of you lab environment:

```
mkdir ~/lab && cd "$_"

git clone https://github.com/gpe-mw-training/ocp-service-mesh-foundations

cd ocp-service-mesh-foundations
```

# 2. Deploy Catalog service version 1

You start by deploying the catalog service to RHOCP. The sidecar proxy is automatically injected.

1. In your terminal window, change to the following directory:

```
cd ~/lab/ocp-service-mesh-foundations/catalog
```

2. Deploy the `catalog` service:

```
oc create \
     -f kubernetes/catalog-service-template.yml \
     -n $OCP_TUTORIAL_PROJECT
```

**Sample Output**

```
deployment.extensions/catalog-v1 created
```

3. Create an OpenShift service entry for the `catalog` service:

```
oc create \
     -f ~/lab/ocp-service-mesh-foundations/catalog/kubernetes/Service.yml \
     -n $OCP_TUTORIAL_PROJECT
```

**Sample Output**

```
service/catalog created
```

4. Monitor the deployment of the pods:

```
oc get pods -w
```

5. Wait until the Ready column displays `2/2` pods and the Status column displays `Running`:

   **Sample Output**

   ```
   NAME                            READY     STATUS    RESTARTS   AGE
   catalog-v1-6b576ffcf8-g6b48   2/2       Running   0          1m
   ```

   The pod corresponding to the *catalog-v1* Deployment includes two containers:

   - **catalog**

     Linux container that includes the business functionality

   - **istio-proxy**

     Red Hat Service Mesh *data plane* side-car container that is in communication with the Service Mesh *data plane*.

6. Press **Ctrl+C** to exit.

Because the `catalog` service is at the end of the service chain (`gateway -> partner -> catalog`), it will not be exposed to the outside world via a route.

---

# 3. Deploy Partner service version 1

Next, you deploy the `partner` service to RHOCP.

1. In your terminal window, change to the following directory:

   ```
   cd ~/lab/ocp-service-mesh-foundations/partner
   ```

2. Deploy the `partner` service:

   ```
   oc create \
        -f kubernetes/partner-service-template.yml \
        -n $OCP_TUTORIAL_PROJECT
   ```

   **Sample Output**

   ```
   deployment.extensions/partner-v1 created
   ```

3. Create an OpenShift service entry for the `partner` service:

```
oc create \
     -f ~/lab/ocp-service-mesh-foundations/partner/kubernetes/Service.yml \
     -n $OCP_TUTORIAL_PROJECT
```

**Sample Output**

```
service/partner created
```

4. Monitor the deployment of the pods:

```
oc get pods -w
```

5. Wait until the Ready column displays `2/2` pods and the Status column displays `Running`:

**Sample Output**

```
NAME                          READY     STATUS    RESTARTS   AGE
partner-v1-68b4854c79-s5vnd   2/2       Running   0          2m
...
```

6. Press **Ctrl+C** to exit.

---

# 4. Deploy Gateway Service

Finally, you deploy the gateway service to RHOCP. This completes the list of services:


1. In your terminal window, change to the following directory:

```
cd ~/lab/ocp-service-mesh-foundations/gateway
```

2. Deploy the `gateway` service:

```
oc create \
     -f kubernetes/gateway-service-template.yml \
     -n $OCP_TUTORIAL_PROJECT
```

**Sample Output**

```
deployment.extensions/gateway created
```

3. Create an OpenShift service entry for the `gateway` service:

```
oc create \
     -f ~/lab/ocp-service-mesh-foundations/gateway/kubernetes/Service.yml \
     -n $OCP_TUTORIAL_PROJECT
```

**Sample Output**

```
service/gateway created
```

4. Monitor the deployment of the pods:

```
oc get pods -w
```

5. Wait until the Ready column displays `2/2` pods and the Status column displays `Running`:

**Sample Output**

```
NAME                          READY     STATUS    RESTARTS   AGE
gateway-7b6bb9dcf7-zb8br      2/2       Running   0          1m
...
```

6. Press **Ctrl+C** to exit.

## 4.1. Expose Gateway Service

1. Take a look at the application architecture:

- The **`gateway`** service is the one your users interact with.
- Inbound traffic into the Service Mesh needs to occur via the Istio *ingress-gateway*.

2. Define a Service Mesh *Gateway* and *VirtualService* to allow inbound traffic to your *gateway* service:

```
echo -en "apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ingress-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - '*'
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ingress-gateway
spec:
  hosts:
  - '*'
  gateways:
  - ingress-gateway
  http:
  - match:
    - uri:
        exact: /
    route:
    - destination:
        host: gateway
        port:
          number: 8080
" > $HOME/service-mesh-gw.yaml
```



3. Apply the Service Mesh *Gateway* and *VirtualService* to OpenShift:

```
oc apply -f $HOME/service-mesh-gw.yaml -n $OCP_TUTORIAL_PROJECT
```

4. Retrieve the URL for the `gateway` service via the Istio *ingress-gateway*:

```
echo "export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.host}')" >> ~/.bashrc

source ~/.bashrc

echo $GATEWAY_URL
```

5. Test the `gateway` service:

```
curl $GATEWAY_URL
```

**Sample Output**

```
gateway => partner => catalog v1 from '6b576ffcf8-g6b48': 1
```

- In this example, `6b576ffcf8-g6b48` is the pod running `v1` and `1` is the number of times you hit the endpoint.

Thus far in this lab, the following has occurred:

1. **Pilot *abstract model* of services has been populated**

   The *Pilot* component of the Service Mesh control plane has discovered services in the *user1-tutorial* namespace. The names of these services populate an *abstract model* of diservices services that Pilot maintains. In your lab, the discovered services include: *catalog*, *partner* and *gateway*.

2. ***Data Plane* traffic has been allowed to flow through services defined in Pilot's *abstract model* of services**

   By default, Red Hat Service Mesh configures every Envoy sidecar proxy to accept traffic on all the ports of its associated workload, and to reach every workload in the mesh when forwarding traffic.

This completes the lab for deploying microservices to a service mesh enabled RHOCP.

# 5. References

- Istio Traffic Management Concepts (https://istio.io/docs/concepts/traffic-management/)
- Learn Istio on Red Hat OpenShift (https://learn.openshift.com/servicemesh)

Last updated 2020-02-26 17:17:48 EST