Service Meshes Lab

Craig Dillon
MSc DevOps

Prerequisites

In this lab we will deploy an AKS cluster, install an Istio service mesh and a small test application. In order to do so, you will need some command line tools to interact with the Kubernetes cluster. You will need;

- Azure CLI - https://learn.microsoft.com/en-us/cli/azure/install-azure-cli
- Kubectl - https://kubernetes.io/docs/tasks/tools/
- Helm - https://helm.sh/docs/intro/install/ (optional)

Learning objectives

This lab will cover the following topics;

- Creating an AKS cluster with Istio as the service mesh
- Installing the Istio-cli tool
- Enabling the ingress gateway for external access
- Installing Istio configuration files including config maps and virtual services
- Deploying a demo application
- Creating routing profiles for Istio
- Testing service mesh behaviour
- Monitoring and metrics tools and dashboards

Lab content

Creating the AKS cluster

Let's create some quick environment variables to make life a little easier;

```
export CLUSTER=aks-istio-lab
export RESOURCE_GROUP=aks-istio
export LOCATION=northeurope
export K8S_VERSION=1.29.2
```

AKS has the ability to deploy with Istio as the service mesh, this is still in preview so we need to login to Azure and add preview tools to our AZ Cli tool.

```
az login
az extension add --name aks-preview
az extension update --name aks-preview
az provider register --namespace Microsoft.ContainerService
```

Next you need to create your resource group;

```
az group create --name ${RESOURCE_GROUP} -- ${LOCATION}
```

Next we deploy the AKS cluster to our service mesh using the latest k8s version;

```
az aks create \
--resource-group ${RESOURCE_GROUP} \
--name ${CLUSTER} \
--enable-asm \
--network-plugin azure \
--node-count 3 \
--kubernetes-version $K8S_VERSION \
--generate-ssh-keys
```

Finally, you will need the AKS credentials to begin interacting with the cluster;

```
az aks get-credentials --resource-group ${RESOURCE_GROUP} --name ${CLUSTER}
```

This will load the required credentials directly to the kublectl CLI tool.

Istio will be installed as part of the AKS deployment, we can query this against the cluster and see Istio pods running in the *aks-istio-system* namespace;

```
az aks show --resource-group ${RESOURCE_GROUP} --name ${CLUSTER}  --query
'serviceMeshProfile.mode'
```

Should result in *"Istio"* and running

```
kubectl get pods -n aks-istio-system
```

Should output something similar to;

```
NAME                                READY   STATUS    RESTARTS   AGE
istiod-asm-1-20-78544bcf69-ckml4    1/1     Running   0          2m16s
istiod-asm-1-20-78544bcf69-k4qwt    1/1     Running   0          2m31s
```

Installing Istio CLI tool

Clone the Istio Git repository, change to the directory and add the Istio bin to your path and verifying that it works;

```
curl -L https://git.io/getLatestIstio | sh -
cd istio-1.21.1
export PATH=$PWD/bin:$PATH
istioctl -i aks-istio-system version
```

Should result in something like;

```
client version: 1.21.1
control plane version: 1.20-dev
data plane version: none
```

Lastly, we need to enable the ingress gateway, and confirm that it has an external IP address to bind to;

```
az aks mesh enable-ingress-gateway --resource-group $RESOURCE_GROUP --name
$CLUSTER --ingress-gateway-type external

kubectl get svc aks-istio-ingressgateway-external -n aks-istio-ingress
```

You should expect to see an output like this;

```
NAME                                TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
AGE
aks-istio-ingressgateway-external   LoadBalancer   10.0.18.3     20.13.193.203
15021:30895/TCP,80:30477/TCP,443:32433/TCP   15m
```

Before moving to the next section, pull a few configuration files from my repository;

```
curl -O
https://raw.githubusercontent.com/x00205790/EADesign_lab1/main/istio-
shared-configmap.yaml
```

```
curl -O
https://raw.githubusercontent.com/x00205790/EADesign_lab1/main/bookinfo-vs-
external.yaml

curl -O
https://raw.githubusercontent.com/x00205790/EADesign_lab1/main/bookinfo-
ext-gw.yaml
```

We can install the Istio config map here, to view the current Istio revision in use on the cluster, run;

```
az aks show -n $CLUSTER -g $RESOURCE_GROUP --query 'serviceMeshProfile'
```

Then run,

```
kubectl apply -f istio-shared-configmap.yaml -n aks-istio-system
```

Congratulations, you've just deployed an AKS cluster with Istio as the service mesh and an ingress gateway. Next we need to install an application for the service mesh to interact with.

Installing the Bookinfo application

Istio comes bundled with a few applications to test the service mesh against, for this we will use Bookinfo. Create the namespace for the application, then tag it so Istio will inject the sidecar and finally deploy the application itself from the Istio Github repository;

```
kubectl create ns bookinfo

kubectl label namespace bookinfo istio.io/rev=asm-1-20

kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-
1.21/samples/bookinfo/platform/kube/bookinfo.yaml -n bookinfo

kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-
1.21/samples/bookinfo/networking/destination-rule-all.yaml -n bookinfo
```

Confirm services are running;

```
kubectl get services -n bookinfo

NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
details       ClusterIP   10.0.13.94     <none>        9080/TCP   4m36s
productpage   ClusterIP   10.0.196.12    <none>        9080/TCP   4m36s
ratings       ClusterIP   10.0.159.1     <none>        9080/TCP   4m36s
reviews       ClusterIP   10.0.54.253    <none>        9080/TCP   4m36s
```

Make sure the pods are in a healthy and running state;

```
kubectl get pods -n bookinfo

NAME                              READY   STATUS    RESTARTS   AGE
details-v1-698d88b-87npg          1/1     Running   0          5m14s
productpage-v1-675fc69cf-vlz85    1/1     Running   0          5m13s
ratings-v1-6484c4d9bb-k2msq       1/1     Running   0          5m14s
reviews-v1-5b5d6494f4-r4pmx       1/1     Running   0          5m14s
reviews-v2-5b667bcbf8-44dpp       1/1     Running   0          5m14s
reviews-v3-5b9bd44f4-h9qh4        1/1     Running   0          5m14s
```

Lastly, a quick curl check to make sure the webpage is being served. There is currently no external gateway configured for the application so we will run the curl command internally on the pod to ensure we get the expected response;

```
kubectl exec "$(kubectl get pod -l app=ratings -o
jsonpath='{.items[0].metadata.name}' -n bookinfo)" -n bookinfo -c ratings -
- curl -sS productpage:9080/productpage | grep -o "<title>.*</title>"


<title>Simple Bookstore App</title>
```

Create the external gateway for Istio

Ok, let's get to work on providing an external gateway so we can reach this service from the internet. For this we will need to create an Istio VirtualService to route incoming traffic to different destinations within the service mesh. We pulled the file *bookinfo-vs-external.yaml* and *bookinfo-ext-gw.yaml* so now it is time to apply them.

```
kubectl apply -f bookinfo-vs-external.yaml -n bookinfo
kubectl apply -f bookinfo-ext-gw.yaml -n bookinfo
```

The VirtualService still needs a gateway to define how incoming traffic should be directed into the service mesh which is defined in *bookinfo-ext-gw.yaml*. We now have service mesh routing configured and open to external traffic.

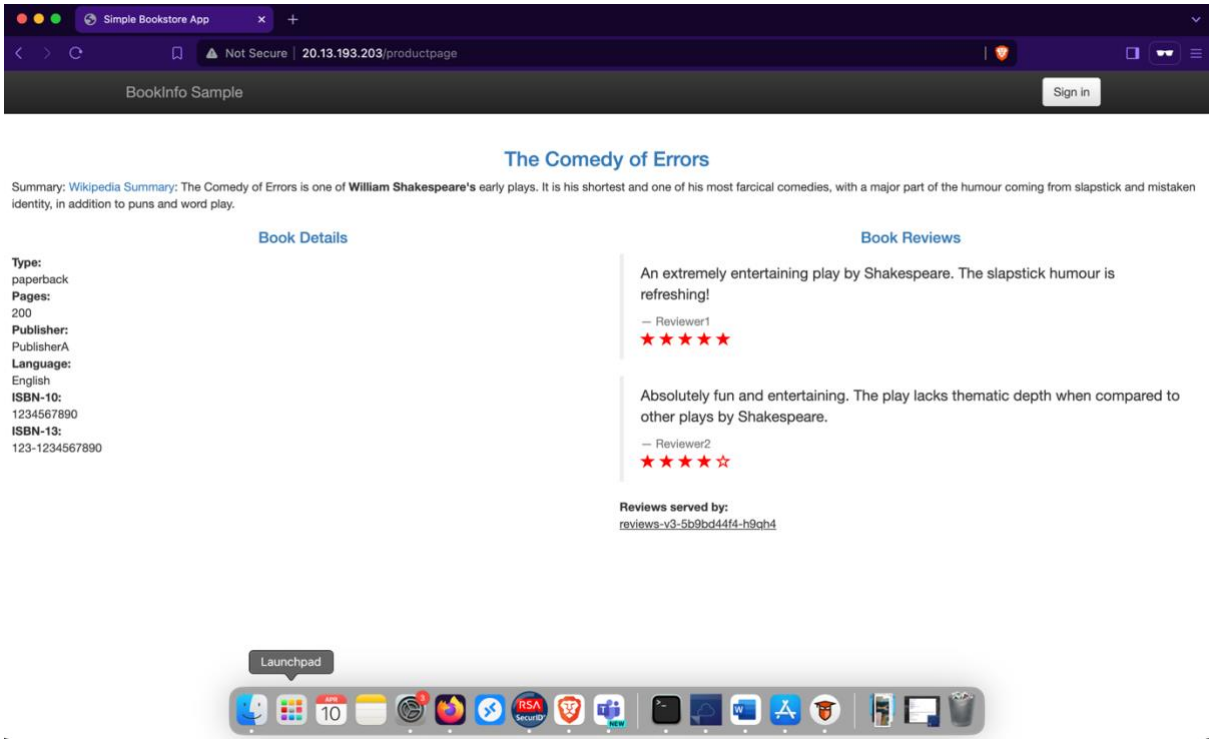
Get the external IP address of the application and test in a web browser;

```
export INGRESS_HOST_EXTERNAL=$(kubectl -n aks-istio-ingress get service
aks-istio-ingressgateway-external -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

export INGRESS_PORT_EXTERNAL=$(kubectl -n aks-istio-ingress get service
aks-istio-ingressgateway-external -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')

export GATEWAY_URL_EXTERNAL=$INGRESS_HOST_EXTERNAL:$INGRESS_PORT_EXTERNAL

echo "http://$GATEWAY_URL_EXTERNAL/productpage"
http://20.13.193.203:80/productpage
```



Observing the service mesh behaviour

Istio bundles some testing scripts to modify the behaviour of the application. If you refresh the page you will see the "**Reviews served by:"** section change, these are the versions of the reviews pod currently serving the page. By applying;

```
kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml -n bookinfo
```

and refreshing the page you should see that it now stays at V1.
Next, apply the following;

```
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml -n bookinfo
```

when refreshing the page, it will remain at v1. However, if you sign in as the user jason with the password jason you will now see it is forced to v2. This is an example of request routing.

Reset the application state by running;

```
kubectl delete -f samples/bookinfo/networking/virtual-service-all-v1.yaml -n bookinfo
```

We can test traffic shifting with weighted subsets for v1 and v3;

```
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml -n bookinfo
```

We can also enforce a HTTP fault on the ratings service for the user jason;

```
kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-abort.yaml -n bookinfo
```

Log in as the jason user and you can see *Ratings service is currently unavailable* where if you log out, ratings are working as expected..

Dashboards and metrics (Optional)

Install some additional packages for monitoring, metrics and dashboards;

```
# Add Prometheus
curl -s https://raw.githubusercontent.com/istio/istio/release-
1.21/samples/addons/prometheus.yaml | sed 's/istio-system/aks-istio-
system/g' | kubectl apply -f -
# Add Grafana
curl -s https://raw.githubusercontent.com/istio/istio/release-
1.21/samples/addons/grafana.yaml | sed 's/istio-system/aks-istio-system/g'
| kubectl apply -f -
# Add Jaeger
curl -s https://raw.githubusercontent.com/istio/istio/release-
1.21/samples/addons/jaeger.yaml | sed 's/istio-system/aks-istio-system/g' |
kubectl apply -f -
# Install Kiali with Helm
helm install \
    --set cr.create=true \
    --set cr.namespace=aks-istio-system \
    --set cr.spec.auth.strategy="anonymous" \
    --namespace aks-istio-system \
    --create-namespace \
    kiali-operator \
    kiali/kiali-operator
```

You will need to open a new terminal for each of these as they run in realtime, or use a service like screen;

```
kubectl port-forward -n aks-istio-system svc/prometheus 9090:9090
kubectl port-forward -n aks-istio-system svc/grafana 3000:3000
# Get the name of the Jaeger pod to start it
kubectl get pods -n aks-istio-system
kubectl port-forward -n aks-istio-system jaeger-{namehere}  16686:16686
kubectl port-forward svc/kiali 20001:20001 -n aks-istio-system
```

Services will be available on localhost;

- Kiali - http://localhost:20001
- Prometheus - http://localhost:9090
- Grafana - http://localhost:3000
- Jaeger - http://localhost:16686

You can generate traffic to explore these applications by running;

```
seq 1 100 | xargs -P 10 -I {} curl -o /dev/null -s -w 'Request: {},
Response: %{http_code}\n' "http://$GATEWAY_URL_EXTERNAL/productpage"
```

Resources

https://learn.microsoft.com/en-us/azure/aks/istio-meshconfig
https://learn.microsoft.com/en-us/azure/aks/istio-deploy-addon
https://istio.io/latest/docs/examples/bookinfo/
https://istio.io/latest/docs/setup/additional-setup/sidecar-injection/
https://kiali.io/docs/installation/installation-guide/install-with-helm/