**Connecting a VM to Istio service mesh**

In this lab, we will learn how to connect a workload running on a virtual machine to the Istio service mesh running on a Kubernetes cluster. The Kubernetes cluster and the virtual machine will be running on the Google Cloud Platform (GCP). We’ll use the single network architecture.

After we’ve created a Kubernetes cluster, we can download, install, and configure Istio.

**Installing Istio on a Kubernetes cluster**

Let’s download Istio 1.10.3:

$ curl -L https://istio.io/downloadIstio | ISTIO\_VERSION=1.10.3 sh -

With Istio downloaded, we can install it using the IstioOperator that sets the mesh ID, cluster name, and network name. Network name will be empty because we’re going with the single network architecture.

Let’s set up a couple of environment variables we’ll use throughout this lab:

export SERVICE\_ACCOUNT="vm-sa"

export VM\_APP="hello-vm"

export VM\_NAMESPACE="vm-namespace"

export WORK\_DIR="${HOME}/vmfiles"

export CLUSTER\_NETWORK=""

export VM\_NETWORK=""

export CLUSTER="Kubernetes"

We can also create the $WORK\_DIR where we’ll store the certificate and other files we’ll have to copy to the virtual machine:

mkdir -p $WORK\_DIR

Next, we’ll initialize the Istio operator and install Istio:

getmesh istioctl operator init

Once we’ve initialized the operator, we can create the IstioOperator resource that specifies the mesh ID, cluster name, and network, and install Istio:

cat <<EOF | kubectl apply -f -

apiVersion: install.istio.io/v1alpha1

kind: IstioOperator

metadata:

name: istio

namespace: istio-system

spec:

values:

global:

meshID: mesh1

multiCluster:

clusterName: "${CLUSTER}"

network: "${CLUSTER\_NETWORK}"

EOF

We’ve mentioned the feature where WorkloadEntry resources get automatically created. This feature is still in active development, so we’ll not be using it in this lab.

We can use the kubectl get iop -n istio-system command to check when the Istio installations’ status changes to HEALTHY.

In the next step, we’ll install the east-west gateway. The control plane uses the east-west gateway to talk to the VM workload and vice-versa.

The gen-eastwest-gateway.sh script is part of the Istio package we downloaded earlier. Change the folder to istio-1.10.3 (or the folder where you’ve unpacked Istio) and run the following command:

samples/multicluster/gen-eastwest-gateway.sh --single-cluster | istioctl install -y -f -

The gen-eastwest-gateway.sh script uses an IstioOperator to deploy an additional gateway called istio-eastwestgateway and configures the service ports.

We can check the new gateway by looking at the Kubernetes services in the istio-system namespace.

Finally, we also need to configure the gateway by exposing the control plane (istiod) through it. We can do that by deploying the expose-istiod.yaml file:

$ kubectl apply -n istio-system -f samples/multicluster/expose-istiod.yaml

gateway.networking.istio.io/istiod-gateway created

virtualservice.networking.istio.io/istiod-vs created

**Preparing virtual machine namespace and files**

We have to create a separate namespace for the virtual machine workloads to store the WorkloadEntry resource and other VM workload-related resources. Additionally, we will have to export the cluster environment file, token, certificate, and other files we will have to transfer to the virtual machine.

We’ll store all files in the $WORK\_DIR we created at the beginning of the lab.

Let’s create the VM namespace and the service account we will use for VM workloads in the same namespace:

$ kubectl create ns "${VM\_NAMESPACE}"

namespace/vm-namespace created

$ kubectl create serviceaccount "${SERVICE\_ACCOUNT}" -n "${VM\_NAMESPACE}"

serviceaccount/vm-sa created

We can now create a WorkloadGroup resource and save it to the workloadgroup.yaml:

cat <<EOF > workloadgroup.yaml

apiVersion: networking.istio.io/v1alpha3

kind: WorkloadGroup

metadata:

name: "${VM\_APP}"

namespace: "${VM\_NAMESPACE}"

spec:

metadata:

labels:

app: "${VM\_APP}"

template:

serviceAccount: "${SERVICE\_ACCOUNT}"

network: "${VM\_NETWORK}"

EOF

The virtual machine needs information about the cluster and Istio’s control plane to connect to it. To generate the required files, we can run getmesh istioctl x workload entry command. We save all generated files to the $WORK\_DIR:

$ getmesh istioctl x workload entry configure -f workloadgroup.yaml -o "${WORK\_DIR}" --clusterID "${CLUSTER}"

Warning: a security token for namespace "vm-namespace" and service account "vm-sa" has been generated and stored at "/vmfiles/istio-token"

configuration generation into directory /vmfiles was successful

**Configuring the virtual machine**

Now it’s time to create and configure a virtual machine. I am running the virtual machine in GCP, just like the Kubernetes cluster. The virtual machine is using the Debian GNU/Linux 10 (Buster) image. Ensure you check “Allow HTTP traffic” under the Firewall section and have SSH access to the instance.

In this example, we run a simple Python HTTP server on port 80. You could configure any other service on a different port. Just make sure you configure the security and firewall rules accordingly.

1. Copy the files from $WORK\_DIR to the home folder on the instance. Replace USERNAME and INSTANCE\_IP accordingly.

$ scp $WORK\_DIR/\* [USERNAME]@[INSTANCE\_IP]:~

Enter passphrase for key '/Users/peterj/.ssh/id\_rsa':

bash: warning: setlocale: LC\_ALL: cannot change locale (en\_US.UTF-8)

cluster.env 100% 589 12.6KB/s 00:00

hosts 100% 38 0.8KB/s 00:00

istio-token 100% 906 19.4KB/s 00:00

mesh.yaml 100% 667 14.4KB/s 00:00

root-cert.pem 100% 1094 23.5KB/s 00:00

Alternatively, you can use use gcloud command and the instance name: gcloud compute scp --zone=us-west1-b ${WORK\_DIR}/\* [INSTANCE\_NAME]:~.

1. SSH into the instance and copy the root certificate to /etc/certs:

sudo mkdir -p /etc/certs

sudo cp root-cert.pem /etc/certs/root-cert.pem

1. Copy the istio-token file to /var/run/secrets/tokens folder:

sudo mkdir -p /var/run/secrets/tokens

sudo cp istio-token /var/run/secrets/tokens/istio-token

1. Download and install the Istio sidecar package:

curl -LO https://storage.googleapis.com/istio-release/releases/1.10.3/deb/istio-sidecar.deb

sudo dpkg -i istio-sidecar.deb

1. Copy cluster.env to /var/lib/istio/envoy/:

sudo cp cluster.env /var/lib/istio/envoy/cluster.env

1. Copy Mesh config (mesh.yaml) to /etc/istio/config/mesh:

sudo cp mesh.yaml /etc/istio/config/mesh

1. Add the istiod host to the /etc/hosts file:

sudo sh -c 'cat $(eval echo ~$SUDO\_USER)/hosts >> /etc/hosts'

1. Change the ownership of files in /etc/certs and /var/lib/istio/envoy to the Istio proxy:

sudo mkdir -p /etc/istio/proxy

sudo chown -R istio-proxy /var/lib/istio /etc/certs /etc/istio/proxy /etc/istio/config /var/run/secrets /etc/certs/root-cert.pem

With all files in place, we can start Istio on the virtual machine:

sudo systemctl start istio

At this point, we have configured the virtual machine to talk with the Istio’s control plane in the Kubernetes cluster.

**Access services from the virtual machine**

Let’s deploy a Hello world application to the Kubernetes cluster. First, we need to enable the automatic sidecar injection in the default namespace:

$ kubectl label namespace default istio-injection=enabled

namespace/default labeled

Next, create the Hello world Deployment and Service.

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-world

labels:

app: hello-world

spec:

replicas: 1

selector:

matchLabels:

app: hello-world

template:

metadata:

labels:

app: hello-world

spec:

containers:

- image: gcr.io/tetratelabs/hello-world:1.0.0

imagePullPolicy: Always

name: svc

ports:

- containerPort: 3000

---

kind: Service

apiVersion: v1

metadata:

name: hello-world

labels:

app: hello-world

spec:

selector:

app: hello-world

ports:

- port: 80

name: http

targetPort: 3000

Save the above file to hello-world.yaml and deploy it using kubectl apply -f hello-world.yaml.

Wait for the Pods to become ready and then go back to the virtual machine and try to access the Kubernetes service:

$ curl http://hello-world.default

Hello World

You can access any service running within your Kubernetes cluster from the virtual machine.

**Run services on the virtual machine**

We can also run a workload on the virtual machine. Switch to the instance and run a simple Python HTTP server:

$ sudo python3 -m http.server 80

Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...

If you try to curl to the instance IP directly, you will get back a response (directory listing):

$ curl [INSTANCE\_IP]

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dt

d">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Directory listing for /</title>

</head>

<body>

<h1>Directory listing for /</h1>

<hr>

...

But what we want to do is to add the workload (Python HTTP service) to the mesh. For that reason, we created the VM namespace earlier. So let’s create a Kubernetes service that represents the VM workload. Note that the name and the label values equal the value of the VM\_APP environment variable we set earlier. Don’t forget to deploy the service to the VM\_NAMESPACE.

apiVersion: v1

kind: Service

metadata:

name: hello-vm

labels:

app: hello-vm

spec:

ports:

- port: 80

name: http-vm

targetPort: 80

selector:

app: hello-vm

Save the above file to hello-vm-service.yaml and deploy it to the VM namespace using kubectl apply -f hello-vm-service.yaml -n ${VM\_NAMESPACE}.

We need to create them manually because we didn’t use the experimental VM auto-registration to create the WorkloadEntry resources automatically.

We need a WorkloadEntry resource that represents the VM workload - the resource uses the VM service account (SERVICE\_ACCOUNT) and the app name in the labels (VM\_APP).

Note that we’ll also need to get the internal IP address of the VM so Istio knows where to reach the VM. Let’s store that in another environment variable (make sure you replace the INSTANCE\_NAME and ZONE with your values):

export VM\_IP=$(gcloud compute instances describe [INSTANCE\_NAME] --format='get(networkInterfaces[0].networkIP)' --zone=[ZONE])

We can now create the WorkloadEntry resource:

cat <<EOF > workloadentry.yaml

apiVersion: networking.istio.io/v1alpha3

kind: WorkloadEntry

metadata:

name: ${VM\_APP}

namespace: ${VM\_NAMESPACE}

spec:

serviceAccount: ${SERVICE\_ACCOUNT}

address: ${VM\_IP}

labels:

app: ${VM\_APP}

instance-id: vm1

EOF

Save the above file to workloadentry.yaml and create then resource in the $VM\_NAMESPACE namespace:

kubectl apply -n ${VM\_NAMESPACE} -f workloadentry.yaml

To “bring” the VM workload inside the mesh, we also need to define the Service entry:

cat <<EOF > serviceentry.yaml

apiVersion: networking.istio.io/v1alpha3

kind: ServiceEntry

metadata:

name: ${VM\_APP}

spec:

hosts:

- ${VM\_APP}

location: MESH\_INTERNAL

ports:

- number: 80

name: http

protocol: HTTP

targetPort: 80

resolution: STATIC

workloadSelector:

labels:

app: ${VM\_APP}

EOF

Note that Istio will automatically create the WorkloadEntry and ServiceEntry in the future.

Create the service entry resource in the $VM\_NAMESPACE:

kubectl apply -n ${VM\_NAMESPACE} -f serviceentry.yaml

We can now use the Kubernetes service name hello-vm.vm-namespace to access the workload on the virtual machine. Let’s run a Pod inside the cluster and try to access the service from there:

$ kubectl run curl --image=radial/busyboxplus:curl -i --tty

If you don't see a command prompt, try pressing enter.

[ root@curl:/ ]$

After you get the command prompt in the Pod, you can run curl and access the workload. You should see a directory listing response. Similarly, you will notice a log entry on the instance where the HTTP server is running:

[ root@curl:/ ]$ curl hello-vm.vm-namespace

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dt

d">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Directory listing for /</title>

</head>

<body>

<h1>Directory listing for /</h1>

<hr>

...

* **hello-vm-service.yaml**178 BYTES

**[DOWNLOADOPENS IN A NEW WINDOW](https://import.cdn.thinkific.com/415508/hellovmservice-210121-161541.yaml" \t "_blank)**

* **workloadgroup.yaml**235 BYTES

**[DOWNLOADOPENS IN A NEW WINDOW](https://import.cdn.thinkific.com/415508/workloadgroup-210121-161541.yaml" \t "_blank)**

* **hello-world.yaml**618 BYTES

**[DOWNLOADOPENS IN A NEW WINDOW](https://import.cdn.thinkific.com/415508/helloworld-210121-161541.yaml" \t "_blank)**

**COMPLETE & CONTINUE**