Installing Anthos Service Mesh on premises

Anthos Service Mesh, which is powered by Istio (https://istio.io/docs/concepts/what-is-istio/), is a framework for connecting, monitoring, and securing services running on Google Kubernetes Engine (GKE) and GKE On-Prem. It lets you create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without requiring any changes in service code. You add Anthos Service Mesh support to services by deploying a sidecar proxy to each of your application's Pods. The sidecar proxy intercepts all network communication between microservices, and is configured and managed using Anthos Service Mesh's control plane functionality.

Note that GKE On-Prem comes with the following Istio components preinstalled:

- Citadel is installed in the `kube-system` namespace.
- Pilot and the Istio Ingress Gateway are installed in the `gke-system` namespace.

GKE On-Prem uses these components to enable ingress and to secure communication between Google-controlled components. If you only need ingress functionality, you don't need to install OSS Istio or Anthos Service Mesh. For more information on configuring ingress, see Enabling ingress (https://cloud.google.com/gke-on-prem/docs/how-to/install-static-ips#enabling_ingress).

When you install Anthos Service Mesh, its components are installed in the `istio-system` namespace. Because the Anthos Service Mesh components are in a different namespace, they don't conflict with the GKE On-Prem preinstalled Istio components.

This guide explains how to install Anthos Service Mesh with the supported core Istio features (https://cloud.google.com/service-mesh/docs/supported-features) enabled on your GKE On-Prem cluster and deploy a demo multi-service application. For information on installing GKE On-

Prem, see Overview of installation (https://cloud.google.com/gke-on-prem/docs/how-to/installation/) in the GKE On-Prem documentation.

Note that the Anthos Service Mesh managed components (https://cloud.google.com/service-mesh/docs/overview#managed_components) aren't currently supported on GKE On-Prem.

Before you begin

Review the following requirements and restrictions before beginning the setup.

Requirements

- Make sure your cluster satisfies the requirements specified in Requirements for Pods and Services (https://istio.io/docs/setup/kubernetes/additional-setup/requirements/).
- Make sure your cluster version is listed in Supported environments (https://cloud.google.com/service-mesh/docs/supported-features#supported_environments). To check your cluster version:

  ```
  gkectl version
  ```

  Output like the following is displayed:

  ```
  1.2.0-gke.6 (git-0912663b0)
  ```

Restrictions

Multiple mesh deployments (https://istio.io/docs/ops/deployment/deployment-models/#mesh-models) in a single Google Cloud project aren't supported.

Setting up your environment

Install the following tools if you don't already have them:

1. Install and initialize the Cloud SDK (https://cloud.google.com/sdk/docs/quickstarts) (the `gcloud` command-line tool).

   If you already have the Cloud SDK installed, make sure to update the components:

   ```
   gcloud components update
   ```

2. Install `kubectl`:

   ```
   gcloud components install kubectl
   ```

Preparing to install Anthos Service Mesh

Before installing Anthos Service Mesh, you need to:

- Set required credentials and permissions.
- Download and extract the Anthos Service Mesh installation file.

Set credentials and permissions

1. Ensure that you have `kubectl` (https://cloud.google.com/anthos/multicluster-management/console/logging-in) for the GKE On-Prem user cluster where you want to install Anthos Service Mesh. Note that you can install Anthos Service Mesh only on a GKE On-Prem user cluster, not an admin cluster.
2. Grant cluster admin permissions to the current user. You need these permissions to create the necessary role based access control (RBAC) (https://cloud.google.com/kubernetes-engine/docs/role-based-access-control) rules for Anthos Service Mesh:

   ```
   kubectl create clusterrolebinding cluster-admin-binding \
     --clusterrole=cluster-admin \
     --user="$(gcloud config get-value core/account)"
   ```

   If you see the `"cluster-admin-binding" already exists` error, you can safely ignore it and continue with the existing cluster-admin-binding.

Download the installation file

LINUX MAC OS WINDOWS

1. Download the Anthos Service Mesh installation file to your current working directory:

```
curl -LO https://storage.googleapis.com/gke-release/asm/istio-1.4.2-asm.2-lin
```

2. Download the signature file and use `openssl` to verify the signature:

```
curl -LO https://storage.googleapis.com/gke-release/asm/istio-1.4.2-asm.2-lin
openssl dgst -verify - -signature istio-1.4.2-asm.2-linux.tar.gz.1.sig istio-
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEWZrGCUaJJr1H8a36sG4UUoXvlXvZ
wQfk16sxprI2gOJ2vFFggdq3ixF2h4qNBt0kI7ciDhgpwS8t+/960IsIgw==
-----END PUBLIC KEY-----
EOF
```

   The expected output is: `Verified OK`

3. Extract the contents of the file to any location on your file system. For example, to extract the contents to the current working directory:

```
tar xzf istio-1.4.2-asm.2-linux.tar.gz
```

   The command creates an installation directory in your current working directory named `istio-1.4.2-asm.2` that contains:

   - Sample applications in `samples`
   - The `istioctl` client binary in the `bin` directory. You can use `istioctl` to manually inject Envoy as a sidecar proxy and to create routing rules and policies.

4. Ensure that you're in the Anthos Service Mesh installation's root directory.

```
cd istio-1.4.2-asm.2
```

5. For convenience, add the `istioctl` client to your PATH:

```
export PATH=$PWD/bin:$PATH
```

Setup namespace and certificate

1. Create a namespace called istio-system:

```
kubectl create namespace istio-system
```

2. Copy the required root certificate into `istio-system` for Citadel. This is required for GKE On-Prem clusters:

```
kubectl get secret istio-ca-secret --namespace=kube-system --export -o yaml | k
```

Installing Anthos Service Mesh

To install Anthos Service Mesh using the ASM configuration profile:

```
istioctl manifest apply --set profile=asm-onprem
```

The ASM profile contains the configuration options suitable for running your services in a production environment. For a full list of features enabled by the ASM profile see Anthos Service Mesh supported features (https://cloud.google.com/service-mesh/docs/supported-features).

The ASM configuration profile:

- Sets your services to run in `PERMISSIVE` mode.

Caution: In `PERMISSIVE` mode, your services can accept both plaintext traffic and mutual TLS traffic at the same time and send plaintext intra-mesh traffic by default. To secure your service mesh, migrate your services to mutual TLS `STRICT` mode, as described in Mutual TLS Migration (https://istio.io/docs/tasks/security/authentication/mtls-migration/).

Customizing the ASM configuration profile

If you need to enable a supported feature that isn't enabled by default in the ASM profile, you can set configuration parameters individually on the command line using `--set values.` For example, to set mTLS to `STRICT` mode by default:

```
istioctl manifest apply --set profile=asm-onprem \
--set values.global.mtls.enabled=true
```

Alternatively, you can specify the configuration in a YAML file and pass it to `istioctl` using the `-f` option.

Note: Whichever approach that you use to customize the ASM configuration profile, make sure the feature that you enable is supported (https://cloud.google.com/service-mesh/docs/supported-features). For more information, see Customizing the configuration (https://istio.io/docs/setup/install/istioctl/#customizing-the-configuration).

Verifying the installation

Check that the control plane Pods in `istio-system` are up:

```
kubectl get pod -n istio-system
```

Expect to see output similar to the following:

```
NAME                                    READY   STATUS    RESTARTS   AGE
istio-citadel-85f4d775cd-dmpj2          1/1     Running   0          18m
istio-galley-5c65896ff7-m2pls           2/2     Running   0          18m
istio-ingressgateway-587cd459f-q6hqt    2/2     Running   0          18m
istio-pilot-9db77b99f-7wfb6             2/2     Running   0          18m
istio-sidecar-injector-69c4d9f875-dt8rn 1/1     Running   0          18m
promsd-55f464d964-lqs7w                 2/2     Running   0          18m
```

Configuring an external IP address

The default Anthos Service Mesh installation assumes that an external IP address is automatically allocated for `LoadBalancer` services. This is not true in GKE On-Prem clusters. Because of this, you need to allocate an IP address manually for the Anthos Service Mesh ingress Gateway resource.

To configure an external IP address, follow one of the sections below, depending on your cluster's load balancing mode:

Integrated load balancing mode

1. Open the `istio-ingressgateway` Service's configuration:

   ```
   kubectl edit svc -n istio-system istio-ingressgateway
   ```

   The configuration for the `istio-ingressgateway` Service opens in your shell's default text editor.

2. In the file, add the following line under the specification (`spec`) block:

   ```
   loadBalancerIP: <your static external IP address>
   ```

   For example:

   ```
   spec:
    loadBalancerIP: 203.0.113.1
   ```

3. Save the file.

Manual load balancing mode

To expose a service of type NodePort with a VIP on your selected load balancer, you need to find out the `nodePort` values first:

1. View the `istio-ingressgateway` Service's configuration in your shell:

   ```
   kubectl get svc -n istio-system istio-ingressgateway -o yaml
   ```

   Each of the ports for Anthos Service Mesh's gateways are displayed. The command output is similar to the following:

   ```
   ...
   ports:
   - name: status-port
     nodePort: 30391
     port: 15020
     protocol: TCP
     targetPort: 15020
   - name: http2
     nodePort: 31380
   ```

```
  port: 80
  protocol: TCP
  targetPort: 80
- name: https
  nodePort: 31390
  port: 443
  protocol: TCP
  targetPort: 443
- name: tcp
  nodePort: 31400
  port: 31400
  protocol: TCP
  targetPort: 31400
- name: https-kiali
  nodePort: 31073
  port: 15029
  protocol: TCP
  targetPort: 15029
- name: https-prometheus
  nodePort: 30253
  port: 15030
  protocol: TCP
  targetPort: 15030
- name: https-grafana
  nodePort: 30050
  port: 15031
  protocol: TCP
  targetPort: 15031
- name: https-tracing
  nodePort: 31204
  port: 15032
  protocol: TCP
  targetPort: 15032
- name: tls
  nodePort: 30158
  port: 15443
  protocol: TCP
  targetPort: 15443
...
```

2. Expose these ports through your load balancer.

   For example, the service port named **`http2`** has **`port`** 80 and **`nodePort`** 31380. Suppose the node addresses for your user cluster are **`192.168.0.10`**, **`192.168.0.11`**, and **`192.168.0.12`**, and your load balancer's VIP is **`203.0.113.1`**.

Configure your load balancer so that traffic sent to `203.0.113.1:80` is forwarded to `192.168.0.10:31380`, `192.168.0.11:31380`, or `192.168.0.12:31380`. You can select the service ports that you want to expose on this given VIP.

Deploying the sample application

Once Istio is installed and all its components are running, you can try deploying one of the sample applications provided with the installation. In this tutorial, we'll install BookInfo (https://istio.io/docs/guides/bookinfo.html). This is a simple mock bookstore application made up of four services that provide a web product page, book details, reviews (with several versions of the review service), and ratings - all managed using Istio. You can find the source code and all the other files used in this example in your Istio installation's samples/bookinfo (https://github.com/istio/istio/tree/master/samples/bookinfo) directory.

Following these steps deploys the BookInfo application's services in an Istio-enabled environment, with Envoy sidecar proxies injected alongside each service to provide Istio functionality.

1. Ensure you're still in the root of the Istio installation directory on your cluster admin machine.
2. Deploy the application using `kubectl apply` and `istioctl kube-inject`. The `kube-inject` command updates the BookInfo deployment so that a sidecar is deployed in each application Pod along with the service.

   ```
   kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/booki
   ```

3. Confirm that the application has been deployed correctly by running the following commands:

   ```
   kubectl get services
   ```

   Output:

   ```
   NAME                       CLUSTER-IP   EXTERNAL-IP   PORT(S)              AGE
   details                    10.0.0.31    <none>        9080/TCP             6m
   kubernetes                 10.0.0.1     <none>        443/TCP              7d
   productpage                10.0.0.120   <none>        9080/TCP             6m
   ```

```
ratings                  10.0.0.15    <none>        9080/TCP             6m
reviews                  10.0.0.170   <none>        9080/TCP             6m
```

and

```
kubectl get pods
```

Output:

```
NAME                                      READY     STATUS    RESTARTS   AGE
details-v1-1520924117-48z17               2/2       Running   0          6m
productpage-v1-560495357-jk1lz            2/2       Running   0          6m
ratings-v1-734492171-rnr5l                2/2       Running   0          6m
reviews-v1-874083890-f0qf0                2/2       Running   0          6m
reviews-v2-1343845940-b34q5               2/2       Running   0          6m
reviews-v3-1813607990-8ch52               2/2       Running   0          6m
```

4. Finally, define the ingress gateway routing for the application:

```
kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

Validating the application deployment

Now that it's deployed, you can see the BookInfo application in action. You already know the external IP address for the ingress gateway as you configured it earlier. So, for example, if you used `203.0.113.1` for your external IP:

```
export GATEWAY_URL=203.0.113.1
```

Trying the application

1. Check that the BookInfo app is running with `curl`:

```
curl -I http://${GATEWAY_URL}/productpage
```

If the response shows `200`, it means the application is working properly with Istio.

2. Now point your browser to `http://$GATEWAY_URL/productpage` to view the BookInfo web page. If you refresh the page several times, you should see different versions of reviews shown in the product page, presented in a round robin style (red stars, black stars, no stars), since we haven't yet used Istio to control the version routing.

Deploying your own application

If you want to try deploying one of your own applications, just follow the same procedure with your own YAML deployment: Istio requires no changes to the application itself. Note that the application must use HTTP/1.1 or HTTP/2.0 protocol for all its HTTP traffic because the Envoy proxy doesn't support HTTP/1.0: it relies on headers that aren't present in HTTP/1.0 for routing.

You can either use `kube-inject` to add the sidecars when deploying the application, as in our example, or enable Istio's automatic sidecar injection (https://cloud.google.com/istio/docs/istio-on-gke/installing#enabling_sidecar_injection) for the namespace where your application is running.

Uninstalling

Run the following command to uninstall the Anthos Service Mesh components:

```
istioctl manifest generate --set profile=asm-onprem | kubectl delete -f -
```

What's next?

Anthos Service Mesh is powered by Istio. You can learn more about Istio on the Istio documentation site (https://istio.io).

Last updated January 23, 2020.