Community tutorials

CONTRIBUTE A TUTORIAL (/COMMUNITY/TUTORIALS/WRITE)

SEARCH TUTORIALS (/DOCS/OPEN-TUTORIALS)

EDIT ON GITHUB
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/EDIT/MASTER/TUTORIALS/TERRAFORM-ASM-UPGRADE.MD)
REPORT ISSUE
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/ISSUES/NEW?TITLE=ISSUE%20WITH%20TUTORIALS/TERRAFORM-ASM-UPGRADE.MD&BODY=ISSUE%20DESCRIPTION)
PAGE HISTORY
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/COMMITS/MASTER/TUTORIALS/TERRAFORM-ASM-UPGRADE.MD)

Upgrade Anthos Service Mesh on GKE with Terraform

Author(s): @cloud-pharaoh (https://github.com/cloud-pharaoh), Published: 2021-07-28

Amina Mansour | Solutions Architect | Google

Contributed by Google employees.

This tutorial shows you how to install Anthos Service Mesh 1.9 with an in-cluster control plane on a GKE cluster using the GKE Anthos Service Mesh Terraform submodule
(https://github.com/terraform-google-modules/terraform-google-kubernetes-engine/tree/master/modules/asm)
and then upgrade to version 1.10 following the revision upgrade process (*canary* upgrade in Istio).

With a revision-based upgrade, you install a new revision of the control plane alongside the existing control plane. When installing the new version, the script includes a revision label that identifies the new control plane. You then migrate to the new version by setting the same revision label on your workloads and

performing a rolling restart to re-inject the proxies so that they use the new Anthos Service Mesh version and configuration. With this approach, you can monitor the effects of the upgrade on a small fraction of your workloads.

After testing your application, you can migrate all traffic to the new version or roll back the changes. This approach is much safer than doing an in-place upgrade in which new control plane components replace the previous version.

Objectives

- Use the GKE Anthos Service Mesh Terraform submodule to do the following:
    - Create a Virtual Private Cloud (VPC) network.
    - Create a GKE cluster.
    - Install Anthos Service Mesh 1.9.
- Deploy the Online Boutique (/service-mesh/docs/onlineboutique-install-kpt) sample app on an Anthos Service Mesh.
- Upgrade to Anthos Service Mesh 1.10 using the revision upgrade process.
- Clean up or destroy all resources with Terraform.

Costs

This tutorial uses the following Google Cloud products:

- Kubernetes Engine (/kubernetes-engine)
- Anthos Service Mesh (/service-mesh)
- Cloud Storage (/storage)

Use the pricing calculator (/products/calculator) to generate a cost estimate based on your projected usage.

Before you begin

1. Select or create a Google Cloud project (https://console.cloud.google.com/projectselector2).

2. Verify that billing is enabled (/billing/docs/how-to/modify-project) for your project.
3. Enable the required APIs:

```
gcloud services enable \
  cloudresourcemanager.googleapis.com \
  container.googleapis.com
```

4. Set an environment variable for your project ID, replacing `[YOUR_PROJECT_ID]` with your project ID:

```
export PROJECT_ID=[YOUR_PROJECT_ID]
```

5. Set the working project to your project:

```
gcloud config set project ${PROJECT_ID}
```

6. Set other environment variables:

```
export PROJECT_NUM=$(gcloud projects describe ${PROJECT_ID} --f
export CLUSTER_1=gke-central
export CLUSTER_1_ZONE=us-central1-a
export CLUSTER_1_CTX=gke_${PROJECT_ID}_${CLUSTER_1_ZONE}_${CLUS
export WORKLOAD_POOL=${PROJECT_ID}.svc.id.goog
export MESH_ID="proj-${PROJECT_NUM}"
export TERRAFORM_SA="terraform-sa"
export ASM_MAJOR_VERSION=1.9
export ASM_MAJOR_VERSION_UPGRADE=1.10
```

7. Create a **`WORKDIR`** folder:

```
mkdir -p asm-upgrade-tutorial && cd asm-upgrade-tutorial && exp
```

8. Create a **`KUBECONFIG`** file for this tutorial:

```
touch ${WORKDIR}/asm-kubeconfig && export KUBECONFIG=${WORKDIR}
```

Prepare Terraform

1. Create a Google Cloud service account and give it the following roles:

```
gcloud --project=${PROJECT_ID} iam service-accounts create ${TE
  --description="terraform-sa" \
  --display-name=${TERRAFORM_SA}

ROLES=(
  'roles/servicemanagement.admin' \
  'roles/storage.admin' \
  'roles/serviceusage.serviceUsageAdmin' \
  'roles/meshconfig.admin' \
  'roles/compute.admin' \
  'roles/container.admin' \
  'roles/resourcemanager.projectIamAdmin' \
  'roles/iam.serviceAccountAdmin' \
  'roles/iam.serviceAccountUser' \
  'roles/iam.serviceAccountKeyAdmin' \
  'roles/gkehub.admin')
for role in "${ROLES[@]}"
do
  gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member "serviceAccount:${TERRAFORM_SA}@${PROJECT_ID}.iam.gs
  --role="$role"
done
```

2. Create the service account credential JSON key for Terraform:

```
gcloud iam service-accounts keys create \
  ${WORKDIR}/${TERRAFORM_SA}.json \
  --iam-account=${TERRAFORM_SA}@${PROJECT_ID}.iam.gserviceaccou
```

3. Set the Terraform credentials and project ID:

```
export GOOGLE_APPLICATION_CREDENTIALS=${WORKDIR}/${TERRAFORM_SA
export TF_VAR_project_id=${PROJECT_ID}
```

```
export TF_VAR_project_number=${PROJECT_NUM}
```

4. Create a Cloud Storage bucket and the backend resource for the Terraform state file:

```
gsutil mb -p ${PROJECT_ID} gs://${PROJECT_ID}
gsutil versioning set on gs://${PROJECT_ID}

cat <<'EOF' > ${WORKDIR}/backend.tf_tmpl
terraform {
  backend "gcs" {
    bucket  = "${PROJECT_ID}"
    prefix  = "tfstate"
  }
}
EOF

envsubst < ${WORKDIR}/backend.tf_tmpl > ${WORKDIR}/backend.tf
```

Deploy resources with Terraform

In this section, you create and apply Terraform files that define the deployment of a VPC network, GKE cluster, and Anthos Service Mesh.

1. Create the **`main.tf`**, **`variables.tf`**, and **`output.tf`** files:

```
cat <<'EOF' > main.tf_tmpl
data "google_client_config" "default" {}

provider "kubernetes" {
  host                   = "https://${module.gke.endpoint}"
  token                  = data.google_client_config.default.ac
  cluster_ca_certificate = base64decode(module.gke.ca_certifica
}

module "vpc" {
  source  = "terraform-google-modules/network/google"
  version = "~> 3.0"

  project_id   = var.project_id
  network_name = var.network
  routing_mode = "GLOBAL"
```

```
  subnets = [
    {
      subnet_name   = var.subnetwork
      subnet_ip     = var.subnetwork_ip_range
      subnet_region = var.region
    }
  ]

  secondary_ranges = {
    (var.subnetwork) = [
      {
        range_name    = var.ip_range_pods
        ip_cidr_range = var.ip_range_pods_cidr
      },
      {
        range_name    = var.ip_range_services
        ip_cidr_range = var.ip_range_services_cidr
      }
    ]
  }
}

module "gke" {
  source                 = "terraform-google-modules/kubernete
  project_id             = var.project_id
  name                   = var.cluster_name
  regional               = false
  region                 = var.region
  zones                  = var.zones
  release_channel        = "REGULAR"
  network                = module.vpc.network_name
  subnetwork             = module.vpc.subnets_names[0]
  ip_range_pods          = var.ip_range_pods
  ip_range_services      = var.ip_range_services
  network_policy         = false
  identity_namespace     = "enabled"
  cluster_resource_labels = { "mesh_id" : "proj-${var.project_n
  node_pools = [
    {
      name         = "asm-node-pool"
      autoscaling  = false
      auto_upgrade = true
      # ASM requires minimum 4 nodes and e2-standard-4
      node_count   = 4
      machine_type = "e2-standard-4"
    },
  ]
 }

module "asm" {
```

```
  source = "github.com/terraform-google-modules/terraform-googl
  cluster_name          = module.gke.name
  cluster_endpoint      = module.gke.endpoint
  project_id            = var.project_id
  location              = module.gke.location
  enable_all            = true
  asm_version           = var.asm_version
  managed_control_plane = false
  service_account       = "${TERRAFORM_SA}@${PROJECT_ID}.iam.gs
  key_file              = "${TERRAFORM_SA}.json"
  options               = ["envoy-access-log"]
  skip_validation       = false
  outdir                = "./${module.gke.name}-outdir-${var.as
}
EOF

cat <<'EOF' > variables.tf_tmpl
variable "project_id" {}

variable "project_number" {}

variable "cluster_name" {
  default = "gke-central"
}

variable "region" {
  default = "us-central1"
}

variable "zones" {
  default = ["us-central1-a"]
}

variable "network" {
  default = "asm-vpc"
}

variable "subnetwork" {
  default = "subnet-01"
}

variable "subnetwork_ip_range" {
  default = "10.10.10.0/24"
}

variable "ip_range_pods" {
  default = "subnet-01-pods"
}

variable "ip_range_pods_cidr" {
```

```
  default = "10.100.0.0/16"
}

variable "ip_range_services" {
  default = "subnet-01-services"
}

variable "ip_range_services_cidr" {
  default = "10.101.0.0/16"
}

variable "asm_version" {
  default = "$ASM_MAJOR_VERSION"
}
EOF

cat <<'EOF' > output.tf
output "kubernetes_endpoint" {
  sensitive = true
  value     = module.gke.endpoint
}

output "client_token" {
  sensitive = true
  value     = base64encode(data.google_client_config.default.ac
}

output "ca_certificate" {
  sensitive = true
  value = module.gke.ca_certificate
}

output "service_account" {
  description = "The default service account used for running n
  value       = module.gke.service_account
}
EOF

envsubst < main.tf_tmpl > main.tf
envsubst < variables.tf_tmpl > variables.tf
```

2. Initialize Terraform and apply the configurations:

```
${TERRAFORM_CMD} init
${TERRAFORM_CMD} plan
${TERRAFORM_CMD} apply -auto-approve
```

Access your cluster

1. Connect to the GKE cluster:

```
gcloud container clusters get-credentials ${CLUSTER_1} --zone $
```

   Remember to unset your **KUBECONFIG** variable when you're finished.

Deploy the Online Boutique app

1. Get the Anthos Service Mesh revision number to label the namespace for automatic Anthos Service Mesh proxy sidecar injection:

```
export ASM_REV=$(kubectl --context=${CLUSTER_1_CTX} get pod -n
  -o jsonpath='{.items[0].metadata.labels.istio\.io/rev}')
```

2. Deploy the Online Boutique app to the GKE cluster:

```
kpt pkg get \
  https://github.com/GoogleCloudPlatform/microservices-demo.git
  online-boutique

kubectl --context=${CLUSTER_1_CTX} create namespace online-bout
kubectl --context=${CLUSTER_1_CTX} label namespace online-bouti
kubectl --context=${CLUSTER_1_CTX} -n online-boutique apply -f
```

3. Wait until all Deployments are ready:

```
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fc
```

```
kubectl --context=${CLUSTER_1_CTX} -n online-boutique wait --fo
```

Access the Online Boutique app

Run the following command to get the IP address of the external load balancer:

```
kubectl --context=${CLUSTER_1_CTX} -n istio-system get service istio
```

Upgrading Anthos Service Mesh

The **`revisioned-istio-ingressgateway`** option creates a revisioned **`istio-ingressgateway`** Deployment, which allows you to control when you switch to the new version. If you don't include this option, the script does an in-place upgrade (https://istio.io/latest/docs/setup/upgrade/in-place/) of the **`istio-ingressgateway`**, which entirely replaces the control plane components.

You must also include any custom overlays or options that you applied during the initial installation.

Prepare Terraform to upgrade Anthos Service Mesh

1. Update the **`variables.tf`** file to the new version of Anthos Service Mesh:

```
sed -i s/ASM_MAJOR_VERSION/ASM_MAJOR_VERSION_UPGRADE/ variables
envsubst < variables.tf_tmpl > variables.tf
sed -i 's/\benvoy-access-log\b/&,revisioned-istio-ingressgatewa
sed -i "/module.gke.location/a mode = \"upgrade\"" main.tf
```

2. Apply Terraform:

```
terraform plan
terraform apply -auto-approve
```

Verify the installed Anthos Service Mesh versions

1. Note the revision label that is on **istiod** and the **istio-ingressgateway**:

```
kubectl --context=${CLUSTER_1_CTX} get pod -n istio-system -L i
```

The output is similar to the following:

```
NAME                                          READY   STATU
istio-ingressgateway-64457f47f8-vctxz         1/1     Runni
istio-ingressgateway-64457f47f8-vfwbv         1/1     Runni
istio-ingressgateway-asm-1102-3-5754c948b-crjkb   1/1     Runni
istio-ingressgateway-asm-1102-3-5754c948b-rdkjz   1/1     Runni
istiod-asm-1102-3-767855fcb5-8xp9s            1/1     Runni
istiod-asm-1102-3-767855fcb5-cktpz            1/1     Runni
istiod-asm-196-2-8544879bbd-bfnd6             1/1     Runni
istiod-asm-196-2-8544879bbd-gp25n             1/1     Runni
```

2. Get the upgraded Anthos Service Mesh revision label:

```
export ISTIOD_UPGRADED_POD=$(kubectl --context=${CLUSTER_1_CTX}
export ASM_REV_UPGRADE=$(kubectl --context=${CLUSTER_1_CTX} -n
  -o jsonpath='{.metadata.labels.istio\.io/rev}')
echo $ASM_REV_UPGRADE
```

3. Switch the **istio-ingressgateway** to the new revision:

```
kubectl --context=${CLUSTER_1_CTX} patch service -n istio-syste
  --type='json' -p="[{"op": "replace", "path": "/spec/selector/
```

4. Add the revision label to the **online-boutique** namespace and remove the **istio-injection** label (if it exists):

```
kubectl --context=${CLUSTER_1_CTX} label namespace online-bouti
```

You can ignore **`istio-injection not found`** in the output. If you see that message, it means that the namespace didn't previously have the **`istio-injection`** label.

5. Restart the Pods to trigger re-injection:

```
kubectl --context=${CLUSTER_1_CTX} rollout restart deployment -
```

6. Wait until all Deployments are restarted:

```
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
kubectl --context=${CLUSTER_1_CTX} -n online-boutique rollout s
```

The output is similar to the following:

```
deployment "adservice" successfully rolled out
deployment "checkoutservice" successfully rolled out
deployment "currencyservice" successfully rolled out
deployment "emailservice" successfully rolled out
deployment "frontend" successfully rolled out
deployment "paymentservice" successfully rolled out
deployment "productcatalogservice" successfully rolled out
deployment "shippingservice" successfully rolled out
deployment "cartservice" successfully rolled out
deployment "loadgenerator" successfully rolled out
deployment "recommendationservice" successfully rolled out
```

7. Verify that your Pods are configured to point to the new version of **`istiod`**:

```
kubectl --context=${CLUSTER_1_CTX} get pods -n online-boutique
```

You should see all of the Pods for the Online Boutique app. You can also verify that the workloads are working correctly by revisiting the Online Botique app.

Finalizing or rolling back the upgrade

At this point, you have two options:

- If everything is working as expected, then follow the steps in "Finalize the upgrade" to transition to the new version of `istiod`.
- If there are any problems, then follow the steps in "Roll back to a previous version" to revert to the previous version.

Finalize the upgrade

In this section, you finalize the Anthos Service Mesh upgrade and delete the previous Anthos Service Mesh version artifacts.

You can't roll back to the previous Anthos Service Mesh version after you perform the steps in this section. If you would like to roll back to the previous Anthos Service Mesh version, skip this section and proceed to the next section.

1. Configure the validating webhook to use the new control plane:

```
curl https://raw.githubusercontent.com/GoogleCloudPlatform/anth
sed -e s/ASM_REV/${ASM_REV_UPGRADE}/ istiod-service.yaml_tmpl >
kubectl --context=${CLUSTER_1_CTX} apply -f istiod-service.yaml
```

2. Delete the old `istio-ingressgateway` Deployment:

```
kubectl --context=${CLUSTER_1_CTX} delete deploy -l \
   app=istio-ingressgateway,istio.io/rev=${ASM_REV} -n istio-sy
   --ignore-not-found=true
```

3. Delete the old version of `istiod`:

```
kubectl --context=${CLUSTER_1_CTX} delete \
  Service,Deployment,HorizontalPodAutoscaler,PodDisruptionBudge
```

```
  istiod-${ASM_REV} -n istio-system --ignore-not-found=true
```

4. Remove the old version of the `IstioOperator` configuration:

```
kubectl --context=${CLUSTER_1_CTX} delete IstioOperator install
```

Roll back to a previous version

In this section, you roll back to the previous Anthos Service Mesh version. If you have already performed the steps in the "Finalize the upgrade" section, then you can't roll back.

1. Restart the Pods to trigger re-injection so that the proxies have the previous version:

```
kubectl --context=${CLUSTER_1_CTX} rollout restart deployment -
```

2. Remove the new `istio-ingressgateway` Deployment:

```
kubectl --context=${CLUSTER_1_CTX} delete deploy -l \
  app=istio-ingressgateway,istio.io/rev=$ASM_REV_UPGRADE -n ist
  --ignore-not-found=true
```

3. Remove the new version of `istiod`:

```
kubectl --context=${CLUSTER_1_CTX} delete \
  Service,Deployment,HorizontalPodAutoscaler,PodDisruptionBudge
  istiod-${ASM_REV_UPGRADE} -n istio-system --ignore-not-found=
```

4. Remove the new version of the `IstioOperator` configuration:

```
kubectl --context=${CLUSTER_1_CTX} delete IstioOperator \
  installed-state-${ASM_REV_UPGRADE} -n istio-system
```

The expected output is similar to the following:

```
istiooperator.install.istio.io "installed-state-REVISION" delet
```

5. Revert Terraform values to the previous version of Anthos Service Mesh:

```
sed -i s/ASM_MAJOR_VERSION_UPGRADE/ASM_MAJOR_VERSION/ variables
envsubst < variables.tf_tmpl > variables.tf
sed -i 's/,revisioned-istio-ingressgateway//' main.tf
sed -i 's/mode = \"upgrade\"//' main.tf
```

Clean up

Terraform destroy

Use the **`terraform destory`** command to destroy all Terraform resources:

```
cd ${WORKDIR}
terraform destroy -auto-approve
```

Delete the project

Alternatively, you can delete the project.

Deleting a project has the following consequences:

- If you used an existing project, you'll also delete any other work that you've done in the project.
- You can't reuse the project ID of a deleted project. If you created a custom project ID that you plan to use in the future, delete the resources inside the project instead. This ensures that URLs that use the project ID, such as an **`appspot.com`** URL, remain available.

To delete a project, do the following:

1. In the Cloud Console, go to the Projects page (https://console.cloud.google.com/iam-admin/projects).

2. In the project list, select the project you want to delete and click **Delete**.
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Learn more about community support for Terraform (/docs/terraform#terraform_support_for).
- Learn more about Anthos Service Mesh (/service-mesh).