Start Lab 01:00:00

How to Use a Network Policy on Google Kubernetes Engine

1 hour Free ★★★★☆ Rate Lab

–/100

GSP480

Google Cloud Self-Paced Labs

Chat

Overview

This lab will show you how to improve the security of your Kubernetes Engine by applying fine-grained restrictions to network communication.

The Principle of Least Privilege is widely recognized as an important design consideration in enhancing the protection of critical systems from faults and malicious behavior. It suggests that every component must be able to access **only** the information and resources that are necessary for its legitimate purpose. This document demonstrates how the Principle of Least Privilege can be implemented within the Kubernetes Engine network layer.

Network connections can be restricted at two tiers of your Kubernetes Engine infrastructure. The first, and coarser grained, mechanism is the application of Firewall Rules at the Network, Subnetwork, and Host levels. These rules are applied outside of the Kubernetes Engine at the VPC level.

Chat

While Firewall Rules are a powerful security measure, and Kubernetes enables you to define even finer grained rules via Network Policies. Network Policies are used to limit intra-cluster communication. Network policies do not apply to pods attached to the host's network namespace.

For this lab you will provision a private Kubernetes Engine cluster and a bastion host with which to access it. A bastion host provides a single host that has access to the cluster, which, when combined with a private Kubernetes network, ensures that the cluster isn't exposed to malicious behavior from the internet at large. Bastions are particularly useful when you do not have VPN access to the cloud network.

Within the cluster, a simple HTTP server and two client pods will be provisioned. You will learn how to use a Network Policy and labeling to only allow connections from one of the client pods.

This lab was created by GKE Helmsman engineers to give you a better understanding of GKE Binary Authorization. You can view this demo on Github here

- We encourage any and all to contribute to our assets!

Chat

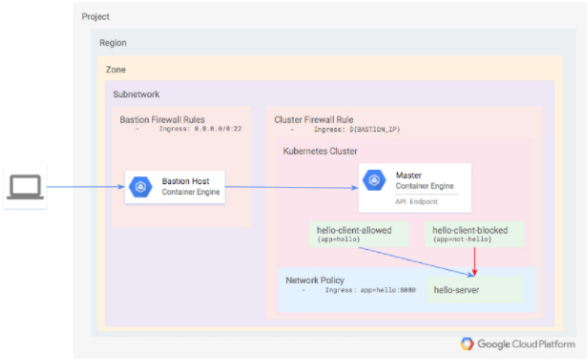
Architecture

You will define a private Kubernetes cluster. Since the cluster is private, neither the API nor the worker nodes will be accessible from the internet. Instead, you will define

a bastion host and use a firewall rule to enable access to it. The bastion's IP address is defined as an authorized network for the cluster, which grants it access to the API.

Within the cluster, provision three workloads:

1. hello-server: this is a simple HTTP server with an internally-accessible endpoint
2. hello-client-allowed: this is a single pod that repeatedly attempts to access hello-server. The pod is labeled such that the Network Policy will allow it to connect to hello-server.
3. hello-client-blocked: this runs the same code as hello-client-allowed but the pod is labeled such that the Network Policy will **not** allow it to connect to hello-server.



Setup

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

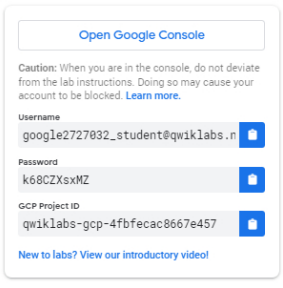
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

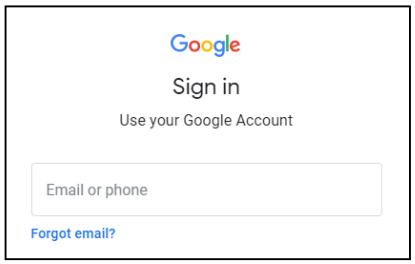
Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

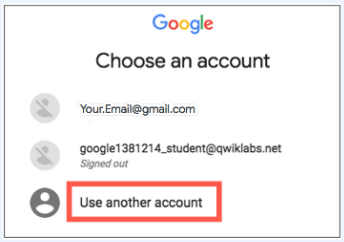


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

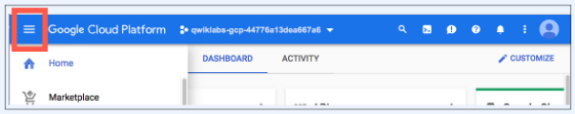
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

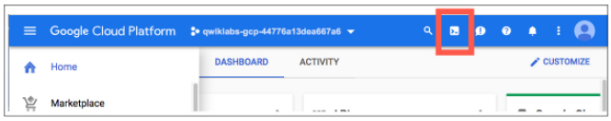
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



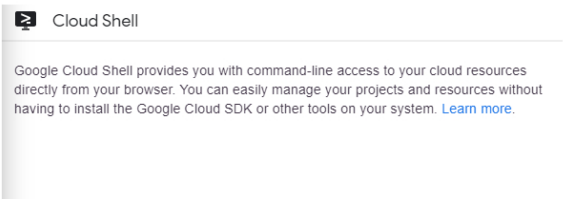
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.

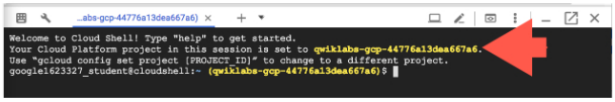


Click **Continue**.



Continue

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]
project = <project_ID>
```

(Example output)

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

Clone demo

Clone the resources needed for this lab by running:

```
git clone https://github.com/GoogleCloudPlatform/gke-network-policy-demo.git
```

Go into the directory for the demo:

```
cd gke-network-policy-demo
```

Set your region and zone

Certain Compute Engine resources live in regions and zones. A region is a specific geographical location where you can run your resources. Each region has one or more zones.

Learn more about regions and zones and see a complete list in Regions & Zones documentation.

Run the following to set a region and zone for your lab (you can use the region/zone that's best for you):

```
gcloud config set compute/region us-central1
```

```
gcloud config set compute/zone us-central1-a
```

Lab setup

This lab will use the following Google Cloud Service APIs, and have been enabled for you:

- `compute.googleapis.com`
- `container.googleapis.com`
- `cloudbuild.googleapis.com`

In addition, the Terraform configuration takes three parameters to determine where the Kubernetes Engine cluster should be created:

- `project ID`
- `region`
- `zone`

For simplicity, these parameters are specified in a file named `terraform.tfvars`, in the `terraform` directory. To ensure the appropriate APIs are enabled and to generate the `terraform/terraform.tfvars` file based on your gcloud defaults, run:

```
make setup-project
```

Type "y" when asked to confirm.

This will enable the necessary Service APIs, and it will also generate a `terraform/terraform.tfvars` file with the following keys. Verify the values themselves will match the output of `gcloud config list` by running:

```
cat terraform/terraform.tfvars
```

Check the latest version for Terraform Google Provider

Replace the terraform provider version with the latest one:

```
sed -i 's/~> 2.10.0/~> 2.14.0/g' terraform/provider.tf
```

Provisioning the Kubernetes Engine Cluster

Next, apply the Terraform configuration within the project root:

```
make tf-apply
```

When prompted, review the generated plan and enter `yes` to deploy the environment.

This will take several minutes to deploy.

Validation

Once complete, Terraform will output a message indicating successful creation of the cluster.

```
...snip...
google_container_cluster.primary: Still creating... (3m0s elapsed)
google_container_cluster.primary: Still creating... (3m10s elapsed)
google_container_cluster.primary: Still creating... (3m20s elapsed)
google_container_cluster.primary: Still creating... (3m30s elapsed)
google_container_cluster.primary: Creation complete after 3m34s (ID: gke-
demo-cluster)

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

You can also confirm the cluster was created successfully by ensuring that Network Policies are enabled for the new cluster.

Verify that `networkPolicyConfig.disabled` is `false` and `networkPolicy.provider` is `CALICO` by running:

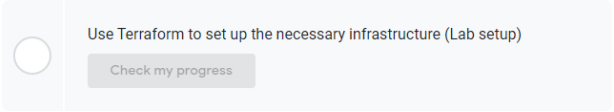
```
gcloud container clusters describe gke-demo-cluster | grep  -A2
networkPolicy
```

(Output)

```
  networkPolicyConfig: {}
clusterIpv4Cidr: 10.0.92.0/22
createTime: '2019-04-18T19:41:27+00:00'
--
networkPolicy:
  enabled: true
  provider: CALICO
```

Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully deployed necessary infrastructure with Terraform, you will see an assessment score.



Now ssh into the bastion for the remaining steps:

```
gcloud compute ssh gke-demo-bastion
```

The newly-created cluster will now be available for the standard `kubectl` commands on the bastion.

Installing the hello server

The test application consists of one simple HTTP server, deployed as `hello-server`, and two clients, one of which will be labeled `app=hello` and the other `app=not-hello`.

All three services can be deployed by applying the hello-app manifests. On the bastion, run:

```
kubectl apply -f ./manifests/hello-app/
```

(Output)

```
deployment.apps/hello-client-allowed created
deployment.apps/hello-client-blocked created
service/hello-server created
deployment.extensions/hello-server created
```

Verify all three pods have been successfully deployed:

```
kubectl get pods
```

You will see one running pod for each of *hello-client-allowed*, *hello-client-blocked*, and *hello-server* deployments.

```
NAME                                  READY     STATUS    RESTARTS
AGE
hello-client-allowed-7d95fcd5d9-t8fsk   |  1/1     Running   0
14m
hello-client-blocked-6497db465d-ckbn8   |  1/1     Running   0
14m
hello-server-7df58f7fb5-nvcvd           |  1/1     Running   0
14m
```

Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully deployed simple HTTP hello server, you will see an assessment score.

Installing the hello server

Check my progress

Chat

Confirming default access to the hello server

First, tail the "allowed" client:

```
kubectl logs --tail 10 -f $(kubectl get pods -oname -l app=hello)
```

Press **Ctrl** + **c** to exit.

Second, tail the logs of the "blocked" client:

```
kubectl logs --tail 10 -f $(kubectl get pods -oname -l app=not-hello)
```

Press **Ctrl** + **c** to exit.

You will notice that both pods are successfully able to connect to the `hello-server` service. This is because you have not yet defined a Network Policy to restrict access. In each of these windows you should see successful responses from the server.

Chat

```
Hostname: hello-server-7df58f7fb5-nvcvd
Hello, world!
Version: 1.0.0
Hostname: hello-server-7df58f7fb5-nvcvd
Hello, world!
Version: 1.0.0
Hostname: hello-server-7df58f7fb5-nvcvd
...
```

Restricting access with a Network Policy

Now you will block access to the `hello-server` pod from all pods that are not labeled with `app=hello`.

The policy definition you'll use is contained in `manifests/network-policy.yaml`

Apply the policy with the following command:

Chat

```
kubectl apply -f ./manifests/network-policy.yaml
```

(Output)

```
networkpolicy.networking.k8s.io/hello-server-allow-from-hello-client
created
```

Tail the logs of the "blocked" client again:

```
kubectl logs --tail 10 -f $(kubectl get pods -oname -l app=not-hello)
```

You'll now see that the output looks like this in the window tailing the "blocked" client:

```
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
wget: download timed out
...
```

Chat

The network policy has now prevented communication to the `hello-server` from

the unlabeled pod.

Restricting namespaces with Network Policies

In the previous example, you defined a network policy that restricts connections based on pod labels. It is often useful to instead label entire namespaces, particularly when teams or applications are granted their own namespaces.

You'll now modify the network policy to only allow traffic from a designated namespace, then you'll move the `hello-allowed` pod into that new namespace.

First, delete the existing network policy:

```
kubectl delete -f ./manifests/network-policy.yaml
```

(Output)

```
networkpolicy.networking.k8s.io "hello-server-allow-from-hello-client"
deleted
```

Create the namespaced version:

```
kubectl create -f ./manifests/network-policy-namespaced.yaml
```

(Output)

```
networkpolicy.networking.k8s.io/hello-server-allow-from-hello-client
created
```

Now observe the logs of the `hello-allowed-client` pod in the default namespace:

```
kubectl logs --tail 10 -f $(kubectl get pods -oname -l app=hello)
```

You will notice it is no longer able to connect to the `hello-server`.

Finally, deploy a second copy of the hello-clients app into the new namespace.

```
kubectl -n hello-apps apply -f ./manifests/hello-app/hello-client.yaml
```

(Output)

```
deployment.apps/hello-client-allowed created
deployment.apps/hello-client-blocked created
```

Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully deployed a second copy of the hello-clients app into the new namespace, you will see an assessment score.

Deploy a second copy of the hello-clients app into the new namespace

Check my progress

Validation

Next, check the logs for the two new `hello-app` clients.

View the logs for the "hello"-labeled app in the app in the `hello-apps` namespace by running:

```
kubectl logs --tail 10 -f -n hello-apps $(kubectl get pods -oname -l
```

```
app=hello -n hello-apps)
```

(Output)

```
Hostname: hello-server-6c6fd59cc9-7fvgp
Hello, world!
Version: 1.0.0
Hostname: hello-server-6c6fd59cc9-7fvgp
Hello, world!
Version: 1.0.0
Hostname: hello-server-6c6fd59cc9-7fvgp
Hello, world!
Version: 1.0.0
Hostname: hello-server-6c6fd59cc9-7fvgp
Hello, world!
Version: 1.0.0
Hostname: hello-server-6c6fd59cc9-7fvgp
```

Both clients are able to connect successfully because *as of Kubernetes 1.10.x NetworkPolicies do not support restricting access to pods within a given namespace.* You can whitelist by pod label, namespace label, or whitelist the union (i.e. OR) of both. But you cannot yet whitelist the intersection (i.e. AND) of pod labels and namespace labels.

Press **Ctrl** + **c** to exit.

Teardown

Qwiklabs will take care of shutting down all the resources used for this lab, but here's what you would need to do to clean up your own environment to save on cost and to be a good cloud citizen:

Log out of the bastion host:

```
exit
```

Run the following to destroy the environment:

```
make teardown
```

(Output)

```
...snip...
google_compute_subnetwork.cluster-subnet: Still destroying... (ID: us-
east1/kube-net-subnet, 20s elapsed)
google_compute_subnetwork.cluster-subnet: Destruction complete after 25s
google_compute_network.gke-network: Destroying... (ID: kube-net)
google_compute_network.gke-network: Still destroying... (ID: kube-net,
10s elapsed)
google_compute_network.gke-network: Still destroying... (ID: kube-net,
20s elapsed)
google_compute_network.gke-network: Destruction complete after 26s

Destroy complete! Resources: 5 destroyed.
```

Troubleshooting in your own environment

The install script fails with a `Permission denied` when running Terraform

The credentials that Terraform is using do not provide the necessary permissions to create resources in the selected projects. Ensure that the account listed in `gcloud config list` has necessary permissions to create resources. If it does, regenerate the application default credentials using `gcloud auth application-default login`.

Invalid fingerprint error during Terraform operations

Terraform occasionally complains about an invalid fingerprint, when updating certain resources. If you see the error below, simply re-run the command.

Congratulations

Chat

Finish Your Quest

This self-paced lab is part of the Qwiklabs Google Kubernetes Engine Best Practices: Security Quest. A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit if you've taken this lab. See other available Qwiklabs Quests.

Take your next lab

Continue your Quest with Securing Applications in Kubernetes Engine - Three Examples, or check out these suggestions:

- Google Kubernetes Engine Security: Binary Authorization
- Using Role-based Access Control in Kubernetes Engine

Next steps / Learn more

- Terraform Google Provider
- Kubernetes Network Policies
- Kubernetes Engine - Creating a Cluster Network Policy
- Kubernetes Engine - Network Policy Tutorial
- Kubernetes Engine - Hardening your cluster's security

Manual Last Updated: June 09, 2020
Lab Last Tested: June 09, 2020

Chat