Start Lab 01:00:00

Using Role-based Access Control in Kubernetes Engine

1 hour Free ★★★★☆ Rate Lab

–/100

Chat

GSP493

Google Cloud Self-Paced Labs

Overview

This lab covers the usage and debugging of role-based access control (RBAC) in a Kubernetes Engine cluster.

While RBAC resource definitions are standard across all Kubernetes platforms, their interaction with underlying authentication and authorization providers needs to be understood when building on any cloud provider.

Chat

RBAC is a powerful security mechanism that provides great flexibility in how you restrict operations within a cluster. This lab will cover two use cases for RBAC:

1. Assigning different permissions to user personas, namely owners and auditors.
2. Granting limited API access to an application running within your cluster.

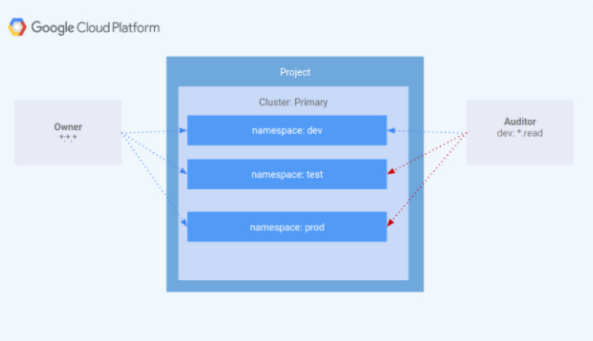
Since RBAC's flexibility can occasionally result in complex rules, common steps for troubleshooting RBAC are included as part of scenario 2.

Architecture

This lab focuses on the use of RBAC within a Kubernetes Engine cluster. It demonstrates how varying levels of cluster privilege can be granted to different user personas. You will provision two service accounts to represent user personas and three namespaces: dev, test, and prod. The "owner" persona will have read-write access to all three namespaces, while the "auditor" persona will have read-only access and be restricted to the dev namespace.

Chat

This lab was created by GKE Helmsman engineers to help you grasp a better understanding of Using role-based access conrols in GKE. You can view this demo on Github here. We encourage any and all to contribute to our assets!



Chat

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.

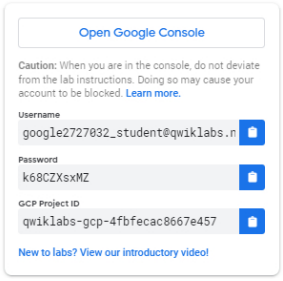
Chat

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.

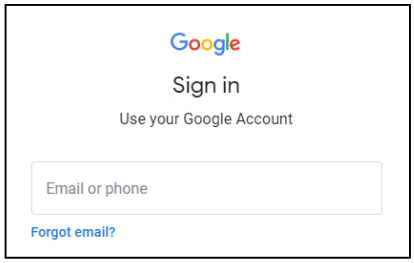
How to start your lab and sign in to the Google Cloud Console

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.



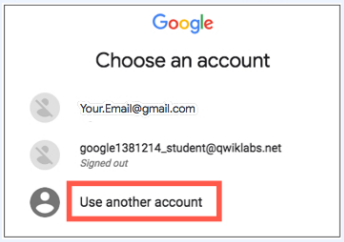
Chat

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.

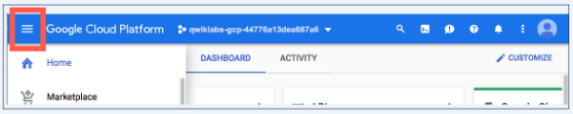
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

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.

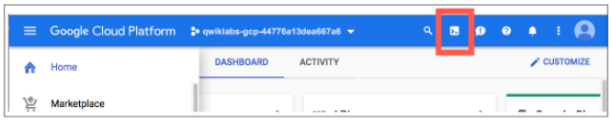
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



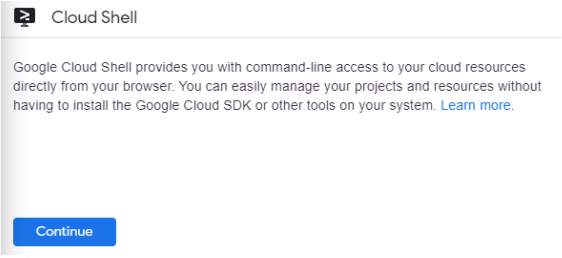
Activate Cloud Shell

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**.



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.

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
```

Clone demo

Clone the resources needed for this lab by running:

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

Go into the directory of the demo:

```
cd gke-rbac-demo
```

Deployment

The steps below will walk you through using Terraform to deploy a Kubernetes

Engine cluster that you will then use for installing test users, applications, and RBAC roles.

Edit the file `gke-rbac-demo/terraform/provider.tf` using the Cloud Shell editor and update the version for Terraform to the latest stable version, 2.12.0:

```
// Configures the default project and zone for underlying Google Cloud
API calls
provider "google" {
  project = var.project
  zone    = var.zone
  version = "~> 2.12.0"
}
```

Once done **save** the file and proceed to the next step.

Provisioning the Kubernetes Engine Cluster

Next, apply the Terraform configuration.

From within the project root, use `make` to apply the Terraform configuration:

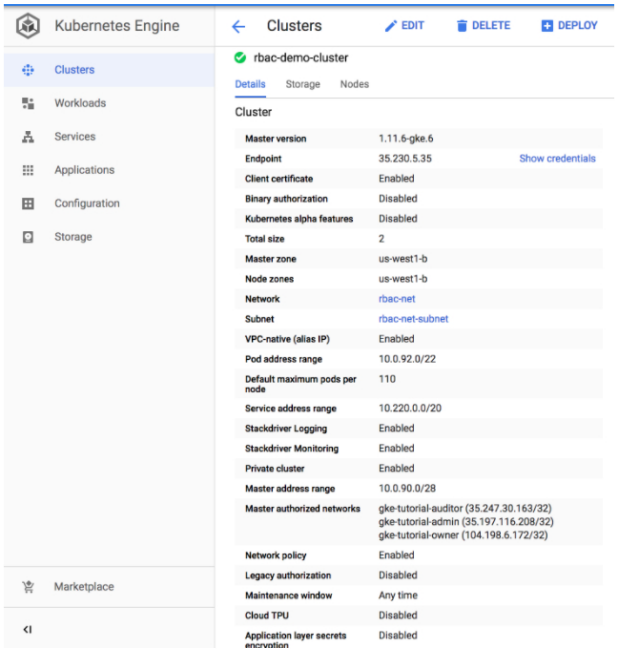
```
make create
```

The setup of this demo takes up to **10-15 minutes**. If there is no error the best thing to do is keep waiting. The execution of `make create` should not be interrupted.

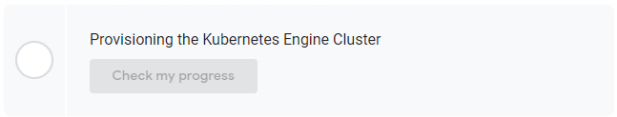
While the resources are building, once you see a `google_compute_instance` get created, you can check on the progress in the Console by going to **Compute Engine** > **VM instances**. Use the **Refresh** button on the VM instances page to view the most up to date information.

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

Confirm the cluster was created successfully in the Console. Go to **Navigation menu** > **Kubernetes Engine** > **Clusters** and click on the cluster that was created. Ensure that Legacy Authorization is disabled for the new cluster.



Click *Check my progress* to verify the objective.



Scenario 1: Assigning permissions by user

persona

IAM - Role

A role named `kube-api-ro-xxxxxxxx` (where `xxxxxxxx` is a random string) has been created with the permissions below as part of the Terraform configuration in `iam.tf`. These permissions are the minimum required for any user that requires access to the Kubernetes API.

- container.apiServices.get
- container.apiServices.list
- container.clusters.get
- container.clusters.getCredentials

Simulating users

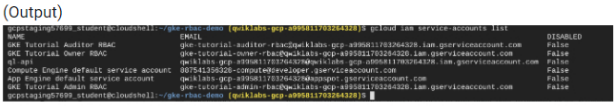
Three service accounts have been created to act as Test Users:

- **admin:** has admin permissions over the cluster and all resources
- **owner:** has read-write permissions over common cluster resources
- **auditor:** has read-only permissions within the dev namespace only

Run the following:

```
gcloud iam service-accounts list
```

(Output)



Three test hosts have been provisioned by the Terraform script. Each node has `kubectl` and `gcloud` installed and configured to simulate a different user persona.

- **gke-tutorial-admin**: kubectl and gcloud are authenticated as a cluster administrator.
- **gke-tutorial-owner**: simulates the 'owner' account
- **gke-tutorial-auditor**: simulates the 'auditor'account

Run the following:

```
gcloud compute instances list
```

(Output)

```
NAME                                                ZONE
MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
rbac-demo-cluster-default-pool-a9cd3468-4vpc    us-central1-a  n1-
standard-1                 10.0.96.5                   RUNNING
rbac-demo-cluster-default-pool-a9cd3468-b47f    us-central1-a  n1-
standard-1                 10.0.96.6                   RUNNING
rbac-demo-cluster-default-pool-a9cd3468-rt5p    us-central1-a  n1-
standard-1                 10.0.96.7                   RUNNING
gke-tutorial-auditor                            us-central1-a  f1-micro
10.0.96.4    35.224.148.28    RUNNING
gke-tutorial-admin                              us-central1-a  f1-micro
10.0.96.3    35.226.237.142   RUNNING
gke-tutorial-owner                              us-central1-a  f1-micro
10.0.96.2    35.194.58.130    RUNNING
```

Creating the RBAC rules

Now you'll create the the Namespaces, Roles, and RoleBindings by logging into the admin instance and applying the `rbac.yaml` manifest.

SSH to the admin:

```
gcloud compute ssh gke-tutorial-admin
```

Create the namespaces, roles, and bindings:

```
kubectl apply -f ./manifests/rbac.yaml
```

(Output)

```
namespace/dev created
namespace/prod created
namespace/test created
role.rbac.authorization.k8s.io/dev-ro created
clusterrole.rbac.authorization.k8s.io/all-rw created
clusterrolebinding.rbac.authorization.k8s.io/owner-binding created
rolebinding.rbac.authorization.k8s.io/auditor-binding created
```

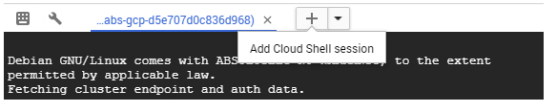
Click *Check my progress* to verify the objective.

Creating the RBAC rules

Check my progress

Managing resources as the owner

Open a new Cloud Shell terminal by clicking the **+** at the top of the terminal window.



You will now SSH into the owner instance and create a simple deployment in each namespace.

SSH to the "owner" instance:

```
gcloud compute ssh gke-tutorial-owner
```

When prompted about the zone, enter `n`, so the default zone is used.

Create a server in each namespace, first `dev`:

```
kubectl create -n dev -f ./manifests/hello-server.yaml
```

(Output)

```
service/hello-server created
deployment.apps/hello-server created
```

And then `prod`:

```
kubectl create -n prod -f ./manifests/hello-server.yaml
```

(Output)

```
service/hello-server created
deployment.apps/hello-server created
```

Then `test`:

```
kubectl create -n test -f ./manifests/hello-server.yaml
```

(Output)

```
service/hello-server created
deployment.apps/hello-server created
```

Click *Check my progress* to verify the objective.

Create server in each namespace

Check my progress

As the owner, you will also be able to view all pods.

On the "owner" instance list all `hello-server` pods in all namespaces by running:

```
kubectl get pods -l app=hello-server --all-namespaces
```

(Output)

```
NAMESPACE   NAME                          READY     STATUS    RESTARTS
AGE
dev         hello-server-6c6fd59cc9-h6zg9   1/1       Running   0
6m
prod        hello-server-6c6fd59cc9-mw2zt   1/1       Running   0
44s
test        hello-server-6c6fd59cc9-sm6bs   1/1       Running   0
39s
```

Viewing resources as the auditor

Now you will open a new terminal, SSH into the auditor instance, and try to view all namespaces.

SSH to the "auditor" instance:

```
gcloud compute ssh gke-tutorial-auditor
```

When prompted about the zone, enter `n`, so the default zone is used.

On the "auditor" instance, list all `hello-server` pods in all namespaces with the following:

```
kubectl get pods -l app=hello-server --all-namespaces
```

You should see an error like the following:

```
Error from server (Forbidden): pods is forbidden: User "gke-tutorial-
auditor@myproject.iam.gserviceaccount.com" cannot list pods at the
cluster scope: Required "container.pods.list" permission
```

The error indicates that you don't have sufficient permissions. The auditor role is restricted to viewing only the resources in the dev namespace, so you'll need to specify the namespace when viewing resources.

Now attempt to view pods in the dev namespace. On the "auditor" instance run the following:

```
kubectl get pods -l app=hello-server --namespace=dev
```

(Output)

```
NAME                          READY     STATUS    RESTARTS   AGE
hello-server-6c6fd59cc9-h6zg9   1/1       Running   0          13m
```

Try to view pods in the test namespace:

```
kubectl get pods -l app=hello-server --namespace=test
```

(Output)

```
Error from server (Forbidden): pods is forbidden: User "gke-tutorial-
auditor@myproject.iam.gserviceaccount.com" cannot list pods in the
namespace "test": Required "container.pods.list" permission.
```

Attempt to view pods in the prod namespace:

```
kubectl get pods -l app=hello-server --namespace=prod
```

(Output)

```
Error from server (Forbidden): pods is forbidden: User "gke-tutorial-
auditor@myproject.iam.gserviceaccount.com" cannot list pods in the
namespace "prod": Required "container.pods.list" permission.
```

Finally, verify the that the auditor has read-only access by trying to create and delete a deployment in the dev namespace.

On the "auditor" instance attempt to create a deployment:

```
kubectl create -n dev -f manifests/hello-server.yaml
```

(Output)

```
Error from server (Forbidden): error when creating "manifests/hello-server.yaml": services is forbidden: User "gke-tutorial-auditor@myproject.iam.gserviceaccount.com" cannot create services in the namespace "dev": Required "container.services.create" permission.
Error from server (Forbidden): error when creating "manifests/hello-server.yaml": deployments.extensions is forbidden: User "gke-tutorial-auditor@myproject.iam.gserviceaccount.com" cannot create deployments.extensions in the namespace "dev": Required "container.deployments.create" permission.
```

On the "auditor" instance, attempt to delete the deployment:

```
kubectl delete deployment -n dev -l app=hello-server
```

(Output)

```
Error from server (Forbidden): deployments.extensions "hello-server" is forbidden: User "gke-tutorial-auditor@myproject.iam.gserviceaccount.com" cannot update deployments.extensions in the namespace "dev": Required "container.deployments.update" permission.
```

Scenario 2: Assigning API permissions to a cluster application

In this scenario you'll go through the process of deploying an application that requires access to the Kubernetes API as well as configure RBAC rules while troubleshooting some common use cases.

Deploying the sample application

The sample application will run as a single pod that periodically retrieves all pods in the default namespace from the API server and then applies a timestamp label to each one.

From the "admin" instance, deploy the `pod-labeler` application. This will also deploy a Role, ServiceAccount, and RoleBinding for the pod:

```
kubectl apply -f manifests/pod-labeler.yaml
```

(Output)

```
role.rbac.authorization.k8s.io/pod-labeler created
serviceaccount/pod-labeler created
rolebinding.rbac.authorization.k8s.io/pod-labeler created
deployment.apps/pod-labeler created
```

Click *Check my progress* to verify the objective.

Deploying the sample application

Check my progress

Diagnosing an RBAC misconfiguration

Now check the status of the pod. Once the container has finished creating, you'll see it error out. Investigate the error by inspecting the pods' events and logs.

On the "admin" instance check the pod status:

```
kubectl get pods -l app=pod-labeler
```

(Output)

```
NAME                          READY     STATUS    RESTARTS   AGE
pod-labeler-6d9757c488-tk6sp   0/1       Error     1          1m
```

On the "admin" instance, view the pod event stream by running:

```
kubectl describe pod -l app=pod-labeler | tail -n 20
```

You should see:

```
Events:
  Type     Reason     Age                    From
Message
  ----     ------     ----                   ----
-------
  Normal   Scheduled  7m35s                  default-scheduler
Successfully assigned default/pod-labeler-5b4bd6cf9-w66jd to gke-rbac-
demo-cluster-default-pool-3d348201-x0pk
  Normal   Pulling    7m34s                  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  pulling image "gcr.io/pso-
examples/pod-labeler:0.1.5"
  Normal   Pulled     6m56s                  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  Successfully pulled image
"gcr.io/pso-examples/pod-labeler:0.1.5"
  Normal   Created    5m29s (x5 over 6m56s)  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  Created container
  Normal   Pulled     5m29s (x4 over 6m54s)  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  Container image "gcr.io/pso-
examples/pod-labeler:0.1.5" already present on machine
  Normal   Started    5m28s (x5 over 6m56s)  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  Started container
  Warning  BackOff    2m25s (x23 over 6m52s)  kubelet, gke-rbac-demo-
cluster-default-pool-3d348201-x0pk  Back-off restarting failed container
```

On the "admin" instance, run the following to check the pod's logs:

```
kubectl logs -l app=pod-labeler
```

(Output)

```
Attempting to list pods
Traceback (most recent call last):
  File "label_pods.py", line 13, in <module>
    ret = v1.list_namespaced_pod("default",watch=False)
  File "build/bdist.linux-
x86_64/egg/kubernetes/client/apis/core_v1_api.py", line 12310, in
list_namespaced_pod
  File "build/bdist.linux-
x86_64/egg/kubernetes/client/apis/core_v1_api.py", line 12413, in
list_namespaced_pod_with_http_info
  File "build/bdist.linux-x86_64/egg/kubernetes/client/api_client.py",
line 321, in call_api
  File "build/bdist.linux-x86_64/egg/kubernetes/client/api_client.py",
line 155, in __call_api
  File "build/bdist.linux-x86_64/egg/kubernetes/client/api_client.py",
line 342, in request
  File "build/bdist.linux-x86_64/egg/kubernetes/client/rest.py", line
231, in GET
  File "build/bdist.linux-x86_64/egg/kubernetes/client/rest.py", line
222, in request
kubernetes.client.rest.ApiException: (403)
Reason: Forbidden
HTTP response headers: HTTPHeaderDict({'Date': 'Fri, 25 May 2018 15:33:15
GMT', 'Audit-Id': 'ae2a0d7c-2ab0-4f1f-bd0f-24107d3c144e', 'Content-
Length': '307', 'Content-Type': 'application/json', 'X-Content-Type-
Options': 'nosniff'})
HTTP response body: {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"pods is forbidden: User
\"system:serviceaccount:default:default\" cannot list pods in the
namespace \"default\": Unknown user
\"system:serviceaccount:default:default\"","reason":"Forbidden","details":
{"kind":"pods"},"code":403}
```

Based on this error, you can see a permissions error when trying to list pods via the API.

The next step is to confirm you are using the correct ServiceAccount.

Fixing the serviceAccountName

By inspecting the pod's configuration, you can see it is using the default ServiceAccount rather than the custom Service Account.

On the "admin" instance, run:

```
kubectl get pod -oyaml -l app=pod-labeler
```

(Output)

```
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext:
```

```
      fsGroup: 9999
      runAsGroup: 9999
      runAsUser: 9999
    serviceAccount: default
```

The `pod-labeler-fix-1.yaml` file contains the fix in the deployment's template spec:

```
      # Fix 1, set the serviceAccount so RBAC rules apply
      serviceAccount: pod-labeler
```

Next you'll apply the fix and view the resulting change.

On the "admin" instance, apply the fix 1 by running:

```
kubectl apply -f manifests/pod-labeler-fix-1.yaml
```

(Output)

```
role.rbac.authorization.k8s.io/pod-labeler unchanged
serviceaccount/pod-labeler unchanged
rolebinding.rbac.authorization.k8s.io/pod-labeler unchanged
deployment.apps/pod-labeler configured
```

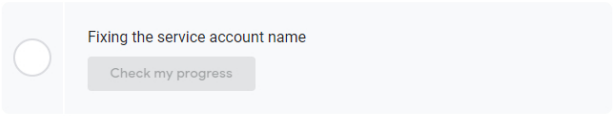
View the change in the deployment configuration:

```
kubectl get deployment pod-labeler -oyaml
```

(Changes in the output)

```
  ...
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: pod-labeler
  ...
```

Click *Check my progress* to verify the objective.



Diagnosing insufficient privileges

Once again, check the status of your pod and you'll notice it is still erring out, but with a different message this time.

On the "admin" instance check the status of your pod:

```
kubectl get pods -l app=pod-labeler
```

(Output)

```
NAME                          READY     STATUS             RESTARTS   AGE
pod-labeler-c7b4fd44d-mr8qh   0/1       CrashLoopBackOff   3          1m
```

You may need to run the previous command again to see this output.

On the "admin" instance, check the pod's logs:

```
kubectl logs -l app=pod-labeler
```

(Output)

```
  File "/usr/local/lib/python3.8/site-
packages/kubernetes/client/rest.py", line 292, in PATCH
    return self.request("PATCH", url,
  File "/usr/local/lib/python3.8/site-
packages/kubernetes/client/rest.py", line 231, in request
    raise ApiException(http_resp=r)
kubernetes.client.rest.ApiException: (403)
Reason: Forbidden
HTTP response headers: HTTPHeaderDict({'Audit-Id': 'f6c67c34-171f-42f3-
b1c9-833e00cedd5e', 'Content-Type': 'application/json', 'X-Content-Type-
Options': 'nosniff', 'Date': 'Mon, 23 Mar 2020 16:35:18 GMT', 'Content-
```

```
Length': '358'})
HTTP response body: {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"pods \"pod-labeler-659c8c99d5-t96pw\" is
forbidden: User \"system:serviceaccount:default:pod-labeler\" cannot
patch resource \"pods\" in API group \"\" in the namespace
\"default\"","reason":"Forbidden","details":{"name":"pod-labeler-
659c8c99d5-t96pw","kind":"pods"},"code":403}
```

Since this is failing on a PATCH operation, you can also see the error in Stackdriver. This is useful if the application logs are not sufficiently verbose.

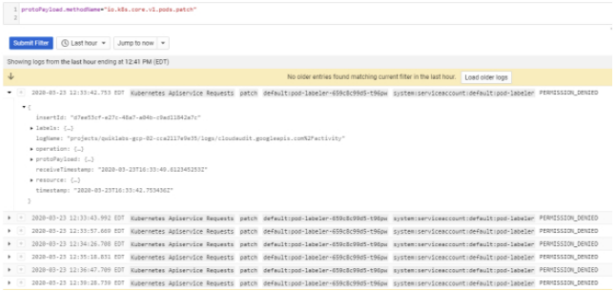
In the Console, select **Navigation menu**, and in the Operations section, click on **Logging**.

Use the dropdown arrow on the Filter field to select **convert to advanced filter**. Replace the existing ling, `resource.type="audited_resource"` with the following:

```
protoPayload.methodName="io.k8s.core.v1.pods.patch"
```

Click **Submit Filter**.

Click on a down arrow next to a log record and explore the details.



Identifying the application's role and permissions

Use the ClusterRoleBinding to find the ServiceAccount's role and permissions.

On the "admin" instance, inspect the `rolebinding` definition:

```
kubectl get rolebinding pod-labeler -oyaml
```

(Output)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"rbac.authorization.k8s.io/v1","kind":"RoleBinding","metadata"
{"annotations":{},"name":"pod-labeler","namespace":"default"},"roleRef":
{"apiGroup":"rbac.authorization.k8s.io","kind":"Role","name":"pod-
labeler"},"subjects":[{"kind":"ServiceAccount","name":"pod-labeler"}]}
  creationTimestamp: "2020-03-23T16:29:05Z"
  name: pod-labeler
  namespace: default
  resourceVersion: "2886"
  selfLink:
/apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/pod-
labeler
  uid: 0e4d5be7-d986-40d0-af1d-a660f9aa4336
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-labeler
subjects:
- kind: ServiceAccount
  name: pod-labeler
```

The `RoleBinding` shows you need to inspect the `pod-labeler` role in the default namespace. Here you can see the role is only granted permission to list pods.

On the "admin" instance, inspect the `pod-labeler` role definition:

```
kubectl get role pod-labeler -oyaml
```

(Output)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```

```
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role","metadata":
{"annotations":{},"name":"pod-labeler","namespace":"default"},"rules":
[{"apiGroups":[""],"resources":["pods"],"verbs":["list"]}]}
  creationTimestamp: "2020-03-23T16:29:05Z"
  name: pod-labeler
  namespace: default
  resourceVersion: "2883"
  selfLink:
/apis/rbac.authorization.k8s.io/v1/namespaces/default/roles/pod-labeler
  uid: c8191869-c7de-42c6-98fc-79a91d9b02a1
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
```

Since the application requires PATCH permissions, you can add it to the "verbs" list of the role, which you will do now.

The `pod-labeler-fix-2.yaml` file contains the fix in it's rules/verbs section:

```
rules:
- apiGroups: [""] # "" refers to the core API group
  resources: ["pods"]
  verbs: ["list","patch"] # Fix 2: adding permission to patch (update)
pods
```

Apply the fix and view the resulting configuration.

On the "admin" instance, apply `fix-2`:

```
kubectl apply -f manifests/pod-labeler-fix-2.yaml
```

(Output)

```
role.rbac.authorization.k8s.io/pod-labeler configured
serviceaccount/pod-labeler unchanged
rolebinding.rbac.authorization.k8s.io/pod-labeler unchanged
deployment.apps/pod-labeler unchanged
```

Click *Check my progress* to verify the objective.

Identifying the application's role and permissions

Check my progress

On the "admin" instance, view the resulting change:

```
kubectl get role pod-labeler -oyaml
```

(Output)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role","metadata":
{"annotations":{},"name":"pod-labeler","namespace":"default"},"rules":
[{"apiGroups":[""],"resources":["pods"],"verbs":["list","patch"]}]}
  creationTimestamp: "2020-03-23T16:29:05Z"
  name: pod-labeler
  namespace: default
  resourceVersion: "8802"
  selfLink:
/apis/rbac.authorization.k8s.io/v1/namespaces/default/roles/pod-labeler
  uid: c8191869-c7de-42c6-98fc-79a91d9b02a1
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
  - patch
```

Because the `pod-labeler` may be in a back-off loop, the quickest way to test the fix is to kill the existing pod and let a new one take it's place.

On the "admin" instance, run the following to kill the existing pod and let the

deployment controller replace it:

```
kubectl delete pod -l app=pod-labeler
```

(Output)

```
pod "pod-labeler-659c8c99d5-t96pw" deleted
```

Verifying successful configuration

Finally, verify the new `pod-labeler` is running and check that the "updated" label has been applied.

On the "admin" instance, list all pods and show their labels:

```
kubectl get pods --show-labels
```

(Output)

```
NAME                           READY     STATUS    RESTARTS   NAME
READY   STATUS    RESTARTS   AGE   LABELS
pod-labeler-659c8c99d5-5qsmw   1/1       Running   0          31s
app=pod-labeler,pod-template-hash=659c8c99d5,updated=1584982487.6428008
```

View the pod's logs to verify there are no longer any errors:

```
kubectl logs -l app=pod-labeler
```

(Output)

```
Attempting to list pods
labeling pod pod-labeler-659c8c99d5-5qsmw
labeling pod pod-labeler-659c8c99d5-t96pw
...
```

Key take-aways

- Container and API server logs will be your best source of clues for diagnosing RBAC issues.
- Use RoleBindings or ClusterRoleBindings to determine which role is specifying the permissions for a pod.
- API server logs can be found in Stackdriver under the Kubernetes resource.
- Not all API calls will be logged to Stackdriver. Frequent, or verbose payloads are omitted by the Kubernetes' audit policy used in Kubernetes Engine. The exact policy will vary by Kubernetes version, but can be found in the open source codebase.

Teardown

When you are finished, and you are ready to clean up the resources that were created, run the following command to remove all resources:

Log out of the bastion host by typing "exit".

Then run the following to destroy the environment:

```
make teardown
```

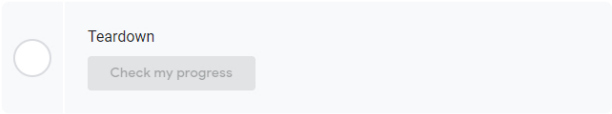
(Output)

```
...snip...
google_service_account.auditor: Destruction complete after 0s
module.network.google_compute_subnetwork.cluster-subnet: Still
destroying... (ID: us-east1/kube-net-subnet, 10s elapsed)
module.network.google_compute_subnetwork.cluster-subnet: Still
destroying... (ID: us-east1/kube-net-subnet, 20s elapsed)
```

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

Destroy complete! Resources: 14 destroyed.
```

Click *Check my progress* to verify the objective.



Chat

Troubleshooting in your own environment

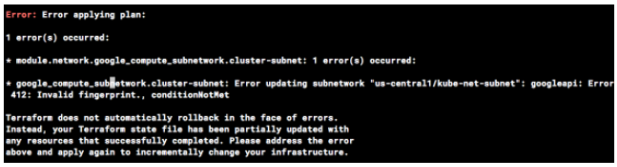
The install script fails with a `Permission denied` when running Terraform

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`.

Chat

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 Google Kubernetes Engine Security: Binary Authorization, or check out these suggestions:

- Hardening Default GKE Cluster Configurations
- Securing Applications in Kubernetes Engine - Three Examples

Next Steps / Learn More

- Kubernetes Engine Role-Based Access Control
- Kubernetes Engine IAM Integration
- Kubernetes Service Account Authentication
- Terraform Documentation

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated April 9, 2020
Lab Last Tested April 9, 2020

Chat