Start Lab 02:15:00

# Gating Deployments with Binary Authorization

Lab 1 hour 30 minutes No cost Introductory



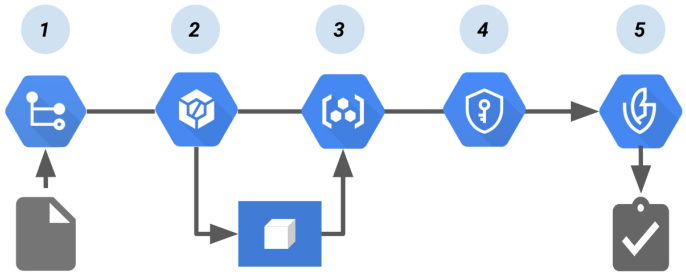
## GSP1183

Google Cloud Self-Paced Labs

## Overview

Binary Authorization is a deploy-time security control that ensures only trusted container images are deployed on Google Kubernetes Engine (GKE) or Cloud Run. With Binary Authorization, you can require images to be signed by trusted authorities during the development process and then enforce signature validation when deploying. By enforcing validation, you can gain tighter control over your container environment by ensuring only verified images are integrated into the build-and-release process.

The following diagram shows the components in a Binary Authorization/Cloud Build setup:




Cloud Build pipeline that creates a Binary Authorization attestation.

In this pipeline:

1. Code to build the container image is pushed to a source repository, such as Cloud Source Repositories.
2. A continuous integration (CI) tool, Cloud Build builds and tests the container.
3. The build pushes the container image to Container Registry or another registry that stores your built images.
4. Cloud Key Management Service, which provides key management for the cryptographic key pair, signs the container image. The resulting signature is then stored in a newly created attestation.
5. At deploy time, the attestor verifies the attestation using the public key from the

key pair. Binary Authorization enforces the policy by requiring signed attestations to deploy the container image.

In this lab you will learn about the tools and techniques to secure deployed artifacts. This lab focuses on artifacts (containers) after they have been created but not deployed to any particular environment.

### What you'll learn

- Image Signing
- Admission Control Policies
- Signing Scanned Images
- Authorizing Signed Images
- Blocked unsigned Images

## Setup and requirements

### 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 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.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

> **Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

> **Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

**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

This 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.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

> **Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

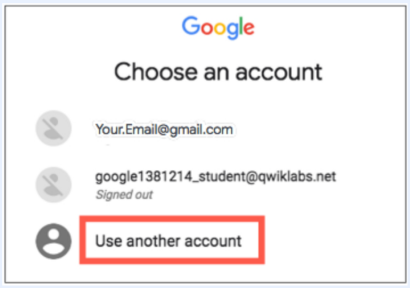
**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

**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


***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 left panel. Then copy and paste the password.

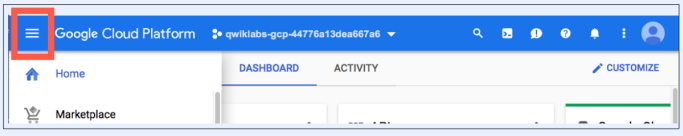
   ***Important:*** You must use the credentials from the left panel. Do not use your Google Cloud Training credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

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.

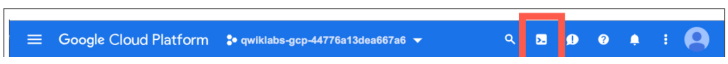
**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.

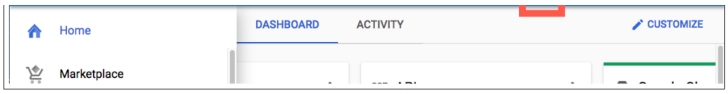



## Activate Cloud Shell

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.



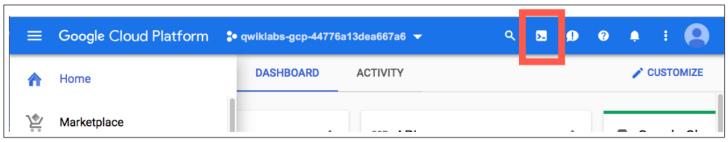



Click **Continue**.


## 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.

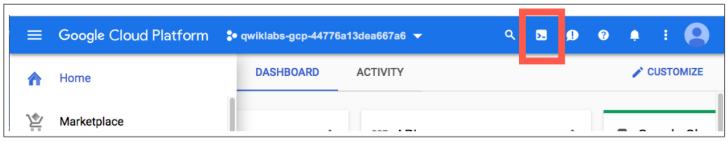



Click **Continue**.

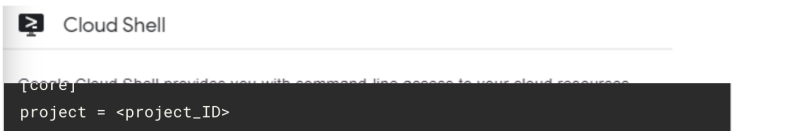

## Activate Cloud Shell

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**.




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

(Example output)

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

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

## Environment Setup

In Cloud Shell, set your project ID and the project number for your project. Save them as `PROJECT_ID` and `PROJECT_NUMBER` variables.

```
export PROJECT_ID=$(gcloud config get-value project)
export PROJECT_NUMBER=$(gcloud projects describe $PROJECT_ID \
    --format='value(projectNumber)')
```

## Enable services

Enable all necessary services:

```
gcloud services enable \
  cloudkms.googleapis.com \
  cloudbuild.googleapis.com \
  container.googleapis.com \
  containerregistry.googleapis.com \
  artifactregistry.googleapis.com \
  containerscanning.googleapis.com \
  ondemandscanning.googleapis.com \
  binaryauthorization.googleapis.com
```

# Task 1. Create Artifact Registry repository

In this lab you will be using Artifact Registry to store and scan your images.

1. Create the repository with the following command:

```
gcloud artifacts repositories create artifact-scanning-repo \
  --repository-format=docker \
  --location="REGION" \
  --description="Docker repository"
```

2. Configure docker to utilize your gcloud credentials when accessing Artifact Registry:

```
gcloud auth configure-docker "REGION"-docker.pkg.dev
```

3. Create and change into a work directory:

```
mkdir vuln-scan && cd vuln-scan
```

4. Next define a sample image. Create a file called `Dockerfile` with the following contents:

```
cat > ./Dockerfile << EOF
FROM python:3.8-alpine

# App
WORKDIR /app
COPY . ./

RUN pip3 install Flask==2.1.0
RUN pip3 install gunicorn==20.1.0
```

```
RUN pip3 install Werkzeug==2.2.2


CMD exec gunicorn --bind :\$PORT --workers 1 --threads 8
main:app

EOF
```

5. Create a file called `main.py` with the following contents:

```
cat > ./main.py << EOF
import os
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    name = os.environ.get("NAME", "Worlds")
    return "Hello {}!".format(name)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0",
port=int(os.environ.get("PORT", 8080)))
EOF
```

6. Use Cloud Build to build and automatically push your container to Artifact Registry.

```
gcloud builds submit . -t "REGION"-docker.pkg.dev/${PROJECT_ID}/art
```

Click **Check my progress** to verify the objective.

Create Artifact Registry repository.

Check my progress

# Task 2. Image Signing

## What is an Attestor

- This person/process is responsible for one link in the chain of trust of the system.
- They hold a cryptographic key, and sign an image if it passes their approval process.
- While the Policy Creator determines policy in a high-level, abstract way, the Attestor is responsible for concretely enforcing some aspect of the policy.
- This can be a real person, like a QA tester or a manager, or a bot in a CI system.
- The security of the system depends on their trustworthiness, so it's important that

Click **Check my progress** to verify the objective.

Create Artifact Registry repository.

Check my progress

# Task 2. Image Signing

### What is an Attestor

- This person/process is responsible for one link in the chain of trust of the system.
- They hold a cryptographic key, and sign an image if it passes their approval process.
- While the Policy Creator determines policy in a high-level, abstract way, the Attestor is responsible for concretely enforcing some aspect of the policy.
- This can be a real person, like a QA tester or a manager, or a bot in a CI system.
- The security of the system depends on their trustworthiness, so it's important that

Click **Check my progress** to verify the objective.

Create Artifact Registry repository.

Check my progress

## Task 2. Image Signing

### What is an Attestor

- This person/process is responsible for one link in the chain of trust of the system.
- They hold a cryptographic key, and sign an image if it passes their approval process.
- While the Policy Creator determines policy in a high-level, abstract way, the Attestor is responsible for concretely enforcing some aspect of the policy.
- This can be a real person, like a QA tester or a manager, or a bot in a CI system.
- The security of the system depends on their trustworthiness, so it's important that

Click **Check my progress** to verify the objective.

Create Artifact Registry repository.

Check my progress

## Task 2. Image Signing

### What is an Attestor

- This person/process is responsible for one link in the chain of trust of the system.
- They hold a cryptographic key, and sign an image if it passes their approval process.
- While the Policy Creator determines policy in a high-level, abstract way, the Attestor is responsible for concretely enforcing some aspect of the policy.
- This can be a real person, like a QA tester or a manager, or a bot in a CI system.
- The security of the system depends on their trustworthiness, so it's important that

Click **Check my progress** to verify the objective.

Create Artifact Registry repository.

Check my progress

# Task 2. Image Signing

## What is an Attestor

- This person/process is responsible for one link in the chain of trust of the system.
- They hold a cryptographic key, and sign an image if it passes their approval process.
- While the Policy Creator determines policy in a high-level, abstract way, the Attestor is responsible for concretely enforcing some aspect of the policy.
- This can be a real person, like a QA tester or a manager, or a bot in a CI system.
- The security of the system depends on their trustworthiness, so it's important that

```
cat > ./vulnz_note.json << EOM
{
  "attestation": {
    "hint": {
      "human_readable_name": "Container Vulnerabilities
attestation authority"
    }
  }
}
EOM
```

2. Store the note

```
NOTE_ID=vulnz_note

curl -vvv -X POST \
    -H "Content-Type: application/json"  \
    -H "Authorization: Bearer $(gcloud auth print-access-token)"
\
    --data-binary @./vulnz_note.json  \

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT_ID}
noteId=${NOTE_ID}"
```

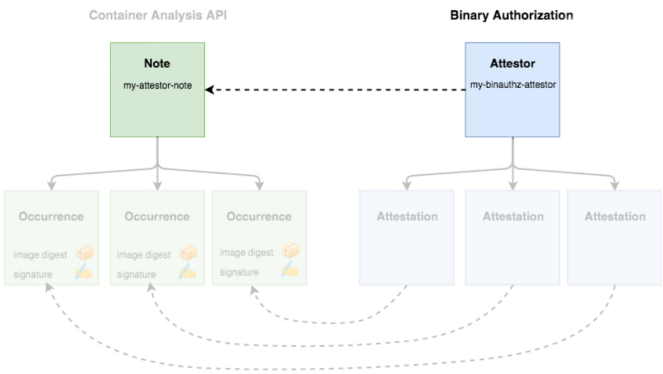
3. Verify the note

```
curl -vvv  \
    -H "Authorization: Bearer $(gcloud auth print-access-token)"
\

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT_ID}
```

Your Note is now saved within the Container Analysis API.

4. Creating an Attestor

Attestors are used to perform the actual image signing process and will attach an occurrence of the note to the image for later verification. To make use of your attestor, you must also register the note with Binary Authorization:




5. Create Attestor

```
ATTESTOR_ID=vulnz-attestor

gcloud container binauthz attestors create $ATTESTOR_ID \
    --attestation-authority-note=$NOTE_ID \
    --attestation-authority-note-project=${PROJECT_ID}
```

6. Verify Attestor

```
gcloud container binauthz attestors list
```

The last line indicates `NUM_PUBLIC_KEYS: 0` you will provide keys in a later step.

Cloud Build automatically creates the `built-by-cloud-build` attestor in your project when you run a build that generates images. So the above command returns two attestors, `vulnz-attestor` and `built-by-cloud-build`. After images are successfully built, Cloud Build automatically signs and creates attestations for them.

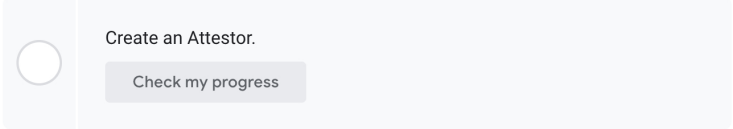
7. The Binary Authorization service account will need rights to view the attestation notes. Provide the access to the IAM Role with the following API call:

```
PROJECT_NUMBER=$(gcloud projects describe "${PROJECT_ID}"  --
format="value(projectNumber)")

BINAUTHZ_SA_EMAIL="service-${PROJECT_NUMBER}@gcp-sa-
binaryauthorization.iam.gserviceaccount.com"


cat > ./iam_request.json << EOM
{
  'resource': 'projects/${PROJECT_ID}/notes/${NOTE_ID}',
  'policy': {
    'bindings': [
      {
        'role':
'roles/containeranalysis.notes.occurrences.viewer',
        'members': [
          'serviceAccount:${BINAUTHZ_SA_EMAIL}'
        ]
      }
    ]
  }
}
EOM
```

8. Use the file to create the IAM Policy:

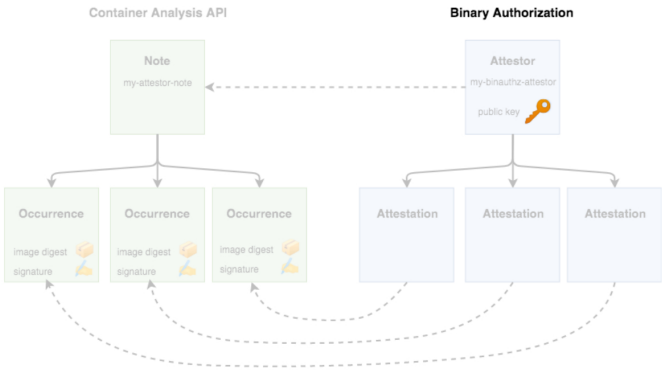
```
curl -X POST  \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $(gcloud auth print-access-token)"
\
    --data-binary @./iam_request.json \

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT_ID}
```

Click **Check my progress** to verify the objective.



# Task 3. Adding a KMS key

Before you can use this attestor, your authority needs to create a cryptographic key pair that can be used to sign container images. This can be done through Google Cloud Key Management Service (KMS).




1. First, add some environment variables to describe the new key:

```
KEY_LOCATION=global
KEYRING=binauthz-keys
KEY_NAME=codelab-key
KEY_VERSION=1
```

2. Create a keyring to hold a set of keys:

```
gcloud kms keyrings create "${KEYRING}" --
location="${KEY_LOCATION}"
```

3. Create a new asymmetric signing key pair for the attestor

```
gcloud kms keys create "${KEY_NAME}" \
    --keyring="${KEYRING}" --location="${KEY_LOCATION}" \
    --purpose asymmetric-signing   \
    --default-algorithm="ec-sign-p256-sha256"
```

You should see your key appear on the KMS page of theCloud console.

4. Now, associate the key with your attestor through the gcloud `binauthz` command:

```
gcloud beta container binauthz attestors public-keys add  \
    --attestor="${ATTESTOR_ID}"  \
    --keyversion-project="${PROJECT_ID}"  \
    --keyversion-location="${KEY_LOCATION}" \
    --keyversion-keyring="${KEYRING}" \
    --keyversion-key="${KEY_NAME}" \
    --keyversion="${KEY_VERSION}"
```

5. If you print the list of authorities again, you should now see a key registered:

```
gcloud container binauthz attestors list
```

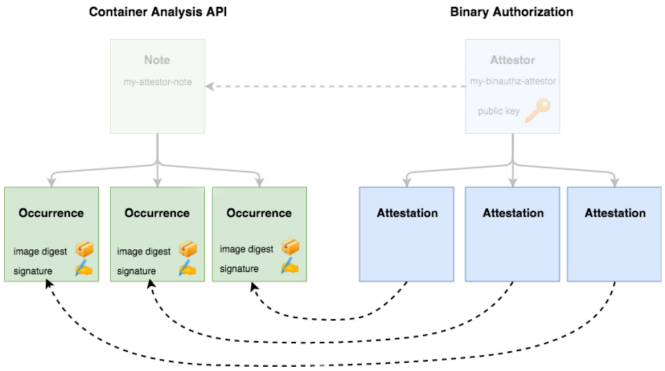
Click **Check my progress** to verify the objective.

Add a KMS key.

Check my progress

## Task 4. Creating a signed attestation

At this point you have the features configured that enable you to sign images. Use the Attestor you created previously to sign the Container Image you've been working with.




An attestation must include a cryptographic signature to state that the attestor has verified a particular container image and is safe to run on your cluster.

1. To specify which container image to attest, run the following to determine its digest:

```
CONTAINER_PATH="REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanr
```

```
DIGEST=$(gcloud container images describe
${CONTAINER_PATH}:latest \
    --format='get(image_summary.digest)')
```

2. Now, you can use gcloud to create your attestation. The command takes in the details of the key you want to use for signing, and the specific container image you want to approve:

```
gcloud beta container binauthz attestations sign-and-create  \
    --artifact-url="${CONTAINER_PATH}@${DIGEST}" \
    --attestor="${ATTESTOR_ID}" \
    --attestor-project="${PROJECT_ID}" \
    --keyversion-project="${PROJECT_ID}" \
    --keyversion-location="${KEY_LOCATION}" \
    --keyversion-keyring="${KEYRING}" \
    --keyversion-key="${KEY_NAME}" \
    --keyversion="${KEY_VERSION}"
```

In Container Analysis terms, this will create a new occurrence, and attach it to your attestor's note.

3. To ensure everything worked as expected, run the following to list your attestations:

```
gcloud container binauthz attestations list \
   --attestor=$ATTESTOR_ID --attestor-project=${PROJECT_ID}
```

# Task 5. Admission control policies

Binary Authorization is a feature in GKE and Cloud Run that provides the ability to validate rules before a container image is allowed to run. The validation executes on any request to run an image be it from a trusted CI/CD pipeline or a user manually trying to deploy an image. This capability allows you to secure your runtime environments more effectively than CI/CD pipeline checks alone.

To understand this capability you will modify the default GKE policy to enforce a strict authorization rule.

1. Create the GKE cluster with binary authorization enabled:

```
gcloud beta container clusters create binauthz \
    --zone "ZONE"  \
    --binauthz-evaluation-mode=PROJECT_SINGLETON_POLICY_ENFORCE
```

2. Allow Cloud Build to deploy to this cluster:

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
        --
member="serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount
\
        --role="roles/container.developer"
```

## Allow All policy

First verify the default policy state and your ability to deploy any image

1. Review existing policy:

```
gcloud container binauthz policy export
```

2. Notice that the enforcement policy is set to `ALWAYS_ALLOW`

`evaluationMode: ALWAYS_ALLOW`

3. Deploy Sample to verify you can deploy anything:

```
kubectl run hello-server --image gcr.io/google-samples/hello-
app:1.0 --port 8080
```

4. Verify the deploy worked:

```
kubectl get pods
```

You will see the following output:

```
NAME           READY   STATUS    RESTARTS   AGE
hello-server   1/1     Running   0          13s
```

5. Delete deployment:

```
kubectl delete pod hello-server
```

## Deny All policy

Now update the policy to disallow all images.

1. Export the current policy to an editable file:

```
gcloud container binauthz policy export  > policy.yaml
```

2. In a text editor, open the `policy.yaml' file and change the evaluationMode `from `ALWAYS_ALLOW` to **`ALWAYS_DENY`**:

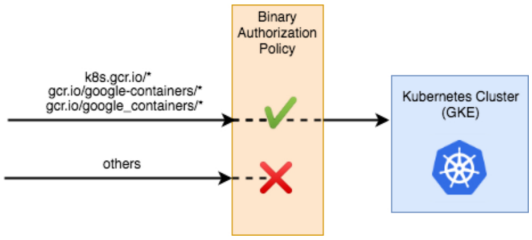
```
edit policy.yaml
```

Ensure the edited policy YAML file appears as follows:

```
globalPolicyEvaluationMode: ENABLE
defaultAdmissionRule:
  evaluationMode: ALWAYS_DENY
  enforcementMode: ENFORCED_BLOCK_AND_AUDIT_LOG
name: projects/PROJECT_ID/policy
```

This policy is relatively simple. The *globalPolicyEvaluationMode* line declares that this policy extends the global policy defined by Google. This allows all official GKE containers to run by default. Additionally, the policy declares a *defaultAdmissionRule* that states that all other pods will be rejected. The admission rule includes an enforcementMode line, which states that all pods that are not conformant to this rule should be blocked from running on the cluster.

For instructions on how to build more complex policies, look through the Binary Authorization documentation.




3. In Cloud Shell run the following to apply the new policy:

```
gcloud container binauthz policy import policy.yaml
```

Wait a few seconds for the change to propagate.

4. Attempt a sample workload deployment:

```
kubectl run hello-server --image gcr.io/google-samples/hello-app:1.0 --port 8080
```

5. Deployment fails with the following message:

```
Error from server (VIOLATES_POLICY): admission webhook "imagepolicywebhook.image-policy.k8s.io" denied the request: Image gcr.io/google-samples/hello-app:1.0 denied by Binary Authorization default admission rule. Denied by always_deny admission rule
```

## Revert the policy to allow all

Before moving on to the next section, revert the policy changes.

1. In a text editor, change the `evaluationMode` from `ALWAYS_DENY` to **ALWAYS_ALLOW**.

```
```

```
edit policy.yaml
```

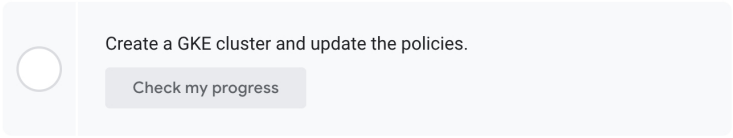
The edited policy YAML file should appear as follows:

```
globalPolicyEvaluationMode: ENABLE
defaultAdmissionRule:
  evaluationMode: ALWAYS_ALLOW
  enforcementMode: ENFORCED_BLOCK_AND_AUDIT_LOG
name: projects/PROJECT_ID/policy
```

2. Apply the reverted policy:

```
gcloud container binauthz policy import policy.yaml
```

Click **Check my progress** to verify the objective.




## Task 6. Automatically signing images

You've enabled image signing, and manually used the Attestor to sign your sample image. In practice you will want to apply attestations during automated processes such as CI/CD pipelines.

In this section you will configure Cloud Build to attest images automatically.

### Roles needed

1. Add Binary Authorization Attestor Viewer role to Cloud Build Service Account:

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member
serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
\
  --role roles/binaryauthorization.attestorsViewer
```

2. Add Cloud KMS CryptoKey Signer/Verifier role to Cloud Build Service Account (KMS-based Signing):

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member
serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
\
  --role roles/cloudkms.signerVerifier
```

3. Add Container Analysis Notes Attacher role to Cloud Build Service Account:

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member
serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
\
  --role roles/containeranalysis.notes.attacher
```

## Provide access for Cloud Build Service Account

Cloud Build will need rights to access the on-demand scanning api.

- Provide access with the following commands:

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
        --
member="serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount
\
        --role="roles/iam.serviceAccountUser"

gcloud projects add-iam-policy-binding ${PROJECT_ID} \
        --
member="serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount
\
        --role="roles/ondemandscanning.admin"
```

## Prepare the Custom Build Cloud Build Step

You'll be using a Custom Build step in Cloud Build to simplify the attestation process. Google provides this Custom Build step which contains helper functions to streamline the process. Before use, the code for the custom build step must be built into a container and pushed to Cloud Build.

- To do this, run the following commands:

```
git clone https://github.com/GoogleCloudPlatform/cloud-builders-
community.git
cd cloud-builders-community/binauthz-attestation
gcloud builds submit . --config cloudbuild.yaml
cd ../..
rm -rf cloud-builders-community
```

## Add a signing step to your `cloudbuild.yaml`

Add the attestation step into your Cloud Build pipeline.

1. Review the signing step below.

**Review only. Do Not Copy**

```
#Sign the image only if the previous severity check passes
- id: 'create-attestation'
  name: 'gcr.io/${PROJECT_ID}/binauthz-attestation:latest'
  args:
    - '--artifact-url'
    - '{{{ project_0.default_region | "REGION" }}}-
docker.pkg.dev/${PROJECT_ID}/artifact-scanning-repo/sample-
image'
    - '--attestor'
    - 'projects/${PROJECT_ID}/attestors/$ATTESTOR_ID'
    - '--keyversion'
    -
'projects/${PROJECT_ID}/locations/$KEY_LOCATION/keyRings/$KEYRING/c
```

2. Write a `cloudbuild.yaml` file with the complete pipeline below:

```
cat > ./cloudbuild.yaml << EOF
steps:
```

```
# build
- id: "build"
  name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', '"REGION"-docker.pkg.dev/${PROJECT_ID}/arti
  waitFor: ['-']

# additional CICD checks (not shown)

#Retag
- id: "retag"
  name: 'gcr.io/cloud-builders/docker'
  args: ['tag',  '"REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-sc


#pushing to artifact registry
- id: "push"
  name: 'gcr.io/cloud-builders/docker'
  args: ['push',  '"REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-s


#Sign the image only if the previous severity check passes
- id: 'create-attestation'
  name: 'gcr.io/${PROJECT_ID}/binauthz-attestation:latest'
  args:
    - '--artifact-url'
    - '"REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanning-repc
    - '--attestor'
    - 'projects/${PROJECT_ID}/attestors/$ATTESTOR_ID'
    - '--keyversion'
    - 'projects/${PROJECT_ID}/locations/$KEY_LOCATION/keyRings/$KEY



images:
  - "REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanning-repo/sa
EOF
```

3. Run the build:

```
gcloud builds submit
```

### Review the build in Cloud Build History

In the Cloud console navigate to **Cloud Build > Build history** page and review that latest build and the successful execution of the build steps.

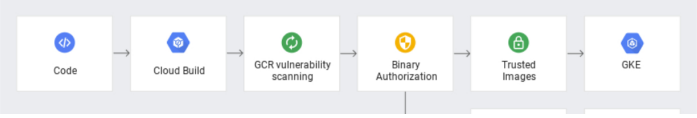
Click **Check my progress** to verify the objective.

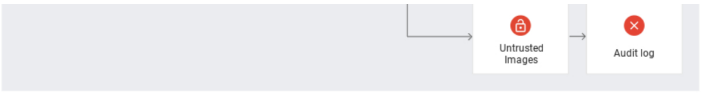
Add a signing step.

Check my progress

## Task 7. Authorizing signed images

Now you will update GKE to use Binary Authorization for validating the image has a signature from the Vulnerability scanning before allowing the image to run.






## Update GKE Policy to Require Attestation

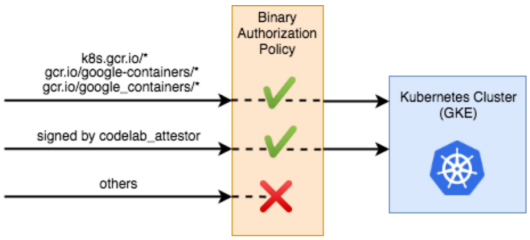
Require images are signed by your Attestor by adding clusterAdmissionRules to your GKE BinAuth Policy

Currently, your cluster is running a policy with one rule: allow containers from official repositories, and reject all others.

1. Overwrite the policy with the updated config using the command below:

```
COMPUTE_ZONE="REGION"

cat > binauth_policy.yaml << EOM
defaultAdmissionRule:
  enforcementMode: ENFORCED_BLOCK_AND_AUDIT_LOG
  evaluationMode: REQUIRE_ATTESTATION
  requireAttestationsBy:
  - projects/${PROJECT_ID}/attestors/vulnz-attestor
globalPolicyEvaluationMode: ENABLE
clusterAdmissionRules:
  ${COMPUTE_ZONE}.binauthz:
    evaluationMode: REQUIRE_ATTESTATION
    enforcementMode: ENFORCED_BLOCK_AND_AUDIT_LOG
    requireAttestationsBy:
    - projects/${PROJECT_ID}/attestors/vulnz-attestor
EOM
```

2. You should now have a new file on disk, called `updated_policy.yaml`. Now, instead of the default rule rejecting all images, it first checks your attestor for verifications.




3. Upload the new policy to Binary Authorization:

```
gcloud beta container binauthz policy import binauth_policy.yaml
```

## Deploy a signed image

1. Get the image digest for the good image:

```
CONTAINER_PATH="REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanr
```

```
DIGEST=$(gcloud container images describe ${CONTAINER_PATH}:good
\
    --format='get(image_summary.digest)')
```

2. Use the digest in the Kubernetes configuration:

```
cat > deploy.yaml << EOM
apiVersion: v1
```

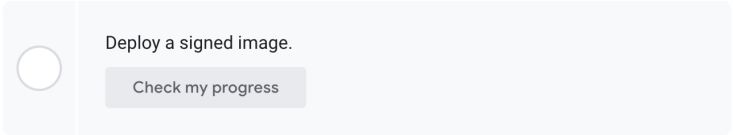
```
kind: Service
metadata:
  name: deb-httpd
spec:
  selector:
    app: deb-httpd
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deb-httpd
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deb-httpd
  template:
    metadata:
      labels:
        app: deb-httpd
    spec:
      containers:
      - name: deb-httpd
        image: ${CONTAINER_PATH}@${DIGEST}
        ports:
        - containerPort: 8080
        env:
          - name: PORT
            value: "8080"

EOM
```

3. Deploy the app to GKE

```
kubectl apply -f deploy.yaml
```

In the Cloud console navigate to **Kubernetes Engine > Workloads** and review the successful deployment of the image.

Click **Check my progress** to verify the objective.




# Task 8. Blocked unsigned Images

## Build an Image

1. Use local docker to build the image to your local cache:

```
docker build -t "REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scar
```

2. Push the unsigned image to the repo:

```
docker push "REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanning
```

3. Get the image digest for the bad image:

```
CONTAINER_PATH="REGION"-docker.pkg.dev/${PROJECT_ID}/artifact-scanr
```

```
DIGEST=$(gcloud container images describe ${CONTAINER_PATH}:bad
\
    --format='get(image_summary.digest)')
```

4. Use the digest in the Kubernetes configuration:

```
cat > deploy.yaml << EOM
apiVersion: v1
kind: Service
metadata:
  name: deb-httpd
spec:
  selector:
    app: deb-httpd
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deb-httpd
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deb-httpd
  template:
    metadata:
      labels:
        app: deb-httpd
    spec:
      containers:
      - name: deb-httpd
        image: ${CONTAINER_PATH}@${DIGEST}
        ports:
        - containerPort: 8080
        env:
          - name: PORT
            value: "8080"

EOM
```

5. Attempt to deploy the app to GKE

```
kubectl apply -f deploy.yaml
```

Review the workload in the console and note the error stating the deployment was denied:

```
No attestations found that were valid and signed by a key trusted by the
attestor
```

Click **Check my progress** to verify the objective.

Deploy an unsigned image.

Check my progress

# Congratulations!

You've learned how to create an Attestor to sign images to validate rules before a container image is allowed to run. You learned how to write a policy to inform Cloud Build to allow or deny access to the GKE cluster, and you have used Binary Authorization with Google Cloud KMS to validate image signatures, and prevent unsigned images access to the Kubernetes cluster.

## What's next:

- Securing image deployments to Cloud Run and Google Kubernetes Engine | Cloud Build Documentation
- Quickstart: Configure a Binary Authorization policy with GKE | Google Cloud

## 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 March 28, 2024**

**Lab Last Tested March 28, 2024**