**1. Introduction**

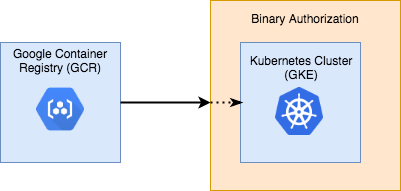
Google Kubernetes Engine and its underlying container model provide increased scalability and manageability for applications hosted in the Cloud. It's easier than ever to launch flexible software applications according to the runtime needs of your system.

This flexibility, however, can come with new challenges. In such environments, it can be difficult to ensure that every component is built, tested, and released according to your best practices and standards, and that only authorized software is deployed to your production environment.

[Binary Authorization (BinAuthz)](https://cloud.google.com/binary-authorization/) is a service that aims to reduce some of these concerns by adding deploy-time policy enforcement to your Kubernetes Engine cluster. Policies can be written to require one or more trusted parties (called "attestors") to approve of an image before it can be deployed. For a multi-stage deployment pipeline where images progress from development to testing to production clusters, attestors can be used to ensure that all required processes have completed before software moves to the next stage.

The identity of attestors is established and verified using cryptographic public keys, and attestations are digitally signed using the corresponding private keys. This ensures that only trusted parties can authorize deployment of software in your environment.

At deployment time, Binary Authorization enforces the policy you defined by checking that the container image has passed all required constraints – including that all required attestors have verified that the image is ready for deployment. If the image passes, the service allows it to be deployed. Otherwise, deployment is blocked and the image cannot be deployed until it is compliant.



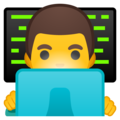
What You'll Build

This codelab describes how to secure a GKE cluster using Binary Authorization. To do this, you will create a policy that all deployments must conform to, and apply it to the cluster. As part of the policy creation, you will create an attestor that can verify container images, and use it to sign and run a custom image.

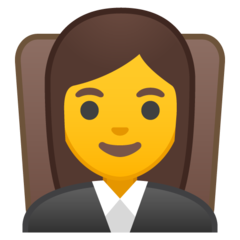
The purpose of this codelab is to give a brief overview of how container signing works with Binary Authorization. With this knowledge, you should feel comfortable building a secure CI/CD pipeline, secured by trusted attestors.

**2. Roles**

Because Binary Authorization concerns the security of your infrastructure, it will typically be interacted with by multiple people with different responsibilities. In this codelab, you will be acting as all of them. Before getting started, it's important to explain the different roles you'll be taking on:

**Deployer**:

* This person/process is responsible for running code on the cluster.
* They aren't particularly concerned with how security guarantees are enforced, that's someone else's job.
* May be a Software Engineer or an automated pipeline.

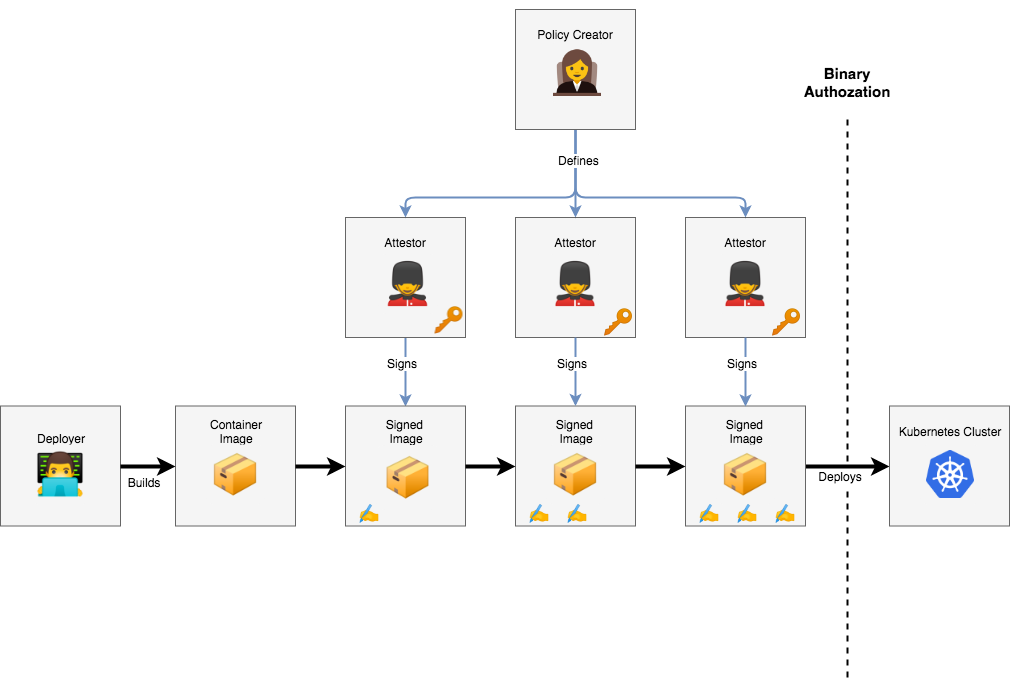
 **Policy Creator:**

* This person is responsible for the big picture security policies of the organization.
* Their job is to make a checklist of rules that must be passed before a container can run.
* They're in charge of the chain of trust, including who needs to sign off an image before it can be considered safe.
* They're not necessarily concerned with the technical details of how to conform to the rules. They might not even know what the software in a container does. They just know about what needs to be done before trust can be established.

**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.
* May be a real person, like a QA tester or a manager, or may be a bot in a CI system.
* The security of the system depends on their trustworthiness, so it's important that their private keys are kept secure.

Each of these roles can represent an individual person, or a team of people in your organization. In a production environment, these roles would likely be managed by separate Google Cloud Platform (GCP) projects, and access to resources would be shared between them in a limited fashion using [Cloud IAM](https://cloud.google.com/iam/).



## 3. Getting Started

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/4426da76922fea23.pngAs a Deployer: |  |

## ****Setting up the Environment****

This codelab can be completed through your web browser using Google Cloud Shell. Click the following link to open a new session:

[Open Google Cloud Shell](https://console.cloud.google.com/cloudshell/editor?shellonly=true)

## ****Setting Your Project****

Our first step is to set the [GCP project](https://cloud.google.com/resource-manager/docs/creating-managing-projects) you want to run the codelab under. You can find a list of the projects under your account with the following command:

gcloud projects list

When you know which project you want to use, set it in an environment variable so you can use it for the rest of the codelab:

PROJECT\_ID=<YOUR\_CHOSEN\_PROJECT\_ID>

gcloud config set project $PROJECT\_ID

## ****Creating a Working Directory****

Through the course of this codelab, you will be creating a few configuration files. You may want to create a new directory to work out of:

mkdir binauthz-codelab ; cd binauthz-codelab

## Enabling the APIs

Before using Binary Authorization, you must enable the relevant APIs on your GCP project:

//enable GKE to create and manage your cluster

gcloud services enable container.googleapis.com

//enable BinAuthz to manage a policy on the cluster

gcloud services enable binaryauthorization.googleapis.com

//enable KMS to manage cryptographic signing keys

gcloud services enable cloudkms.googleapis.com

Alternatively, you can enable the APIs for your project through the [Google Cloud Platform API Library](https://console.cloud.google.com/apis/library/container.googleapis.com).

## ****Setting up a Cluster****

Next, set up a Kubernetes cluster for your [project](https://cloud.google.com/resource-manager/docs/creating-managing-projects) through Kubernetes Engine. The following command will create a new cluster named "binauthz-codelab" in the zone us-central1-a with binary authorization enabled:

gcloud beta container clusters create \

--enable-binauthz \

--zone us-central1-a \

binauthz-codelab

Note: creating a cluster can take a few minutes to complete

Once your cluster has been created, add it to your local environment so you can interact with it locally using kubectl:

gcloud container clusters get-credentials binauthz-codelab --zone us-central1-a

## Running a Pod

Now, let's add a container to the new cluster. The following command will create a simple Dockerfile you can use:

cat << EOF > Dockerfile

FROM alpine

CMD tail -f /dev/null

EOF

This container will do nothing but run the "tail -f /dev/null" command, which will cause it to wait forever. It's not a particularly useful container, but it will allow you to test the security of your cluster.

Build the container and push it to [Google Container Registry (GCR)](https://cloud.google.com/container-registry/):

#set the GCR path you will use to host the container image

CONTAINER\_PATH=us.gcr.io/${PROJECT\_ID}/hello-world

#build container

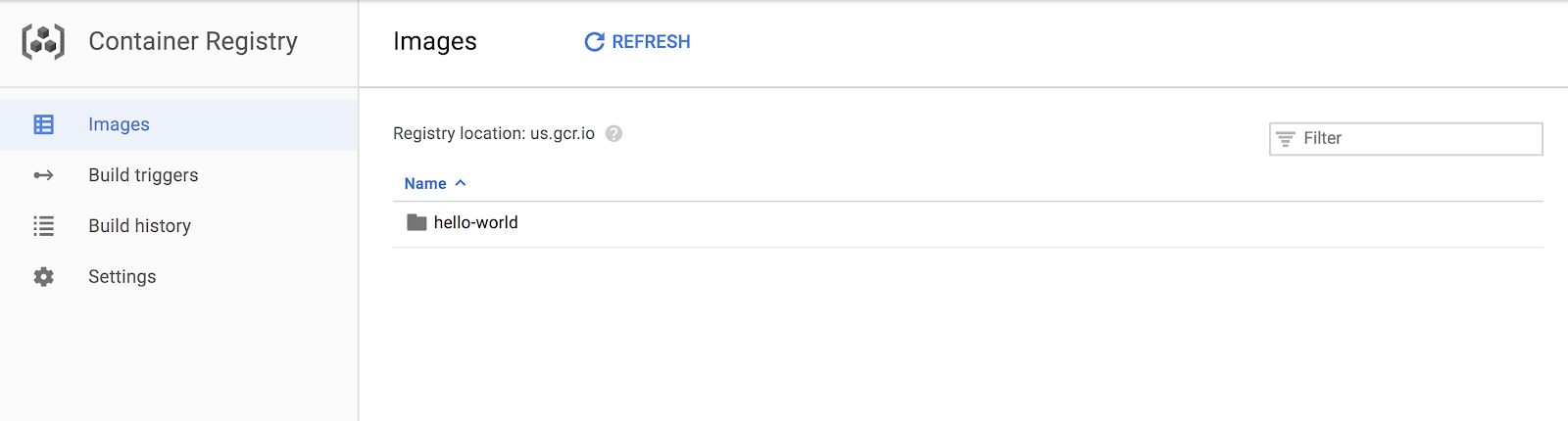
docker build -t $CONTAINER\_PATH ./

#push to GCR

gcloud auth configure-docker --quiet

docker push $CONTAINER\_PATH

You should now be able to see the newly created container in the [Container Registry web interface](https://console.cloud.google.com/gcr).



Now, run the container on your cluster:

kubectl create deployment hello-world --image=$CONTAINER\_PATH

If everything worked well, your container should be silently running.

You can verify this by listing the running pods:

kubectl get pods

## 4. Securing the Cluster with a Policy

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/5b1748abb8d8b699.pngAs a Policy Creator: |  |

## ****Adding a Policy****

You now have a cluster set up and running your code. Now, secure the cluster with a policy.

The first step is to create a policy file:

cat > ./policy.yaml << EOM

globalPolicyEvaluationMode: ENABLE

defaultAdmissionRule:

evaluationMode: ALWAYS\_DENY

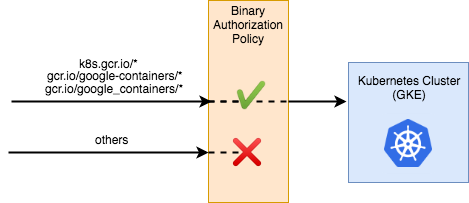
enforcementMode: ENFORCED\_BLOCK\_AND\_AUDIT\_LOG

EOM

This policy is relatively simple. The [globalPolicyEvaluationMode](https://cloud.google.com/binary-authorization/docs/policy-yaml-reference#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](https://cloud.google.com/binary-authorization/docs/policy-yaml-reference#defaultadmissionrule) that states that all [other pods will be rejected](https://cloud.google.com/binary-authorization/docs/policy-yaml-reference#evaluationmode). The admission rule includes an [enforcementMode](https://cloud.google.com/binary-authorization/docs/policy-yaml-reference#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](https://cloud.google.com/binary-authorization/docs/policy-yaml-reference).

Note: while developing a policy, you may want to make use of [dryrun mode](https://cloud.google.com/binary-authorization/docs/example-policies#allow_deployments_in_dryrun_mode). This mode will allow all pods to run, but will log any policy violation events to the audit log.



Now, you can apply the policy to your project:

gcloud container binauthz policy import policy.yaml

Alternatively, the policy can also be set through the [Google Cloud Console UI](https://console.cloud.google.com/security/binary-authorization).

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/4426da76922fea23.pngAs a Deployer: |  |

## ****Testing the Policy****

Your new policy should prevent any custom container images from being deployed on the cluster. You can verify this by deleting your pod and attempting to run it again:

kubectl delete deployment --all

kubectl delete event --all

kubectl create deployment hello-world --image=$CONTAINER\_PATH

If you check the cluster for pods, you should notice that no pods are running this time:

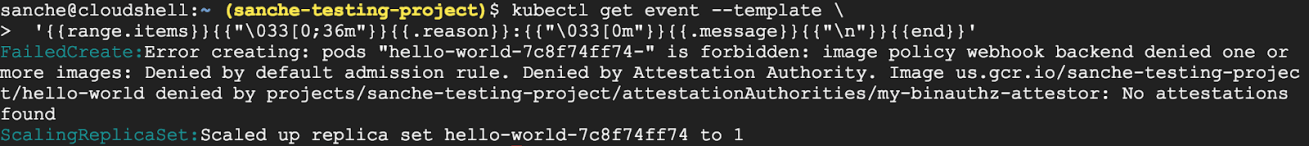
kubectl get pods

You may need to run the command a second time to see the pods disappear. kubectl checked the pod against the policy, found that it didn't conform to the rules, and rejected it.

You can see the rejection listed as a kubectl event:

kubectl get event --template \

'{{range.items}}{{"\033[0;36m"}}{{.reason}}:{{"\033[0m"}}{{.message}}{{"\n"}}{{end}}'



Note: the above command can be run without the template string. The template was added to improve readability of the output

## ****5. Container Analysis Primer****

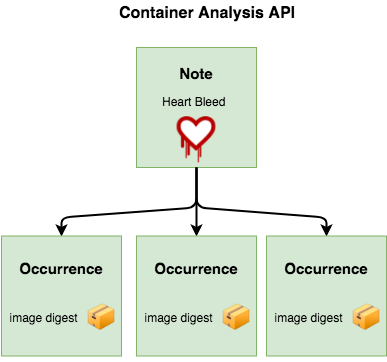
Attestors in Binary Authorization are implemented on top of the [Cloud Container Analysis API](https://cloud.google.com/container-analysis/api/reference/rest/), so it is important to describe how that works before going forward. The Container Analysis API was designed to allow you to associate metadata with specific container images.

## ****Container Analysis Concepts:****

**Note**: represents a piece of metadata in a generalized way. It's associated with the GCP project of whoever created it, not with any particular container image.

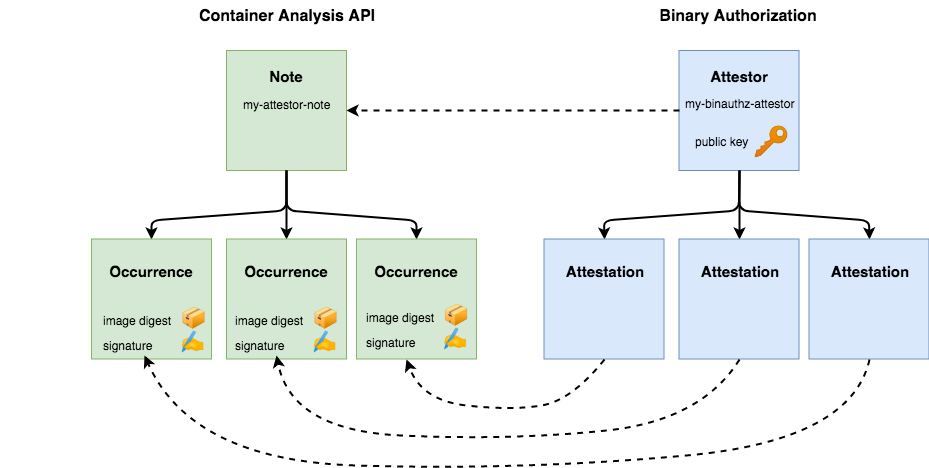
**Occurrence**: represents a single instance of a Note, associated with a specific container. While a Note can describe a vulnerability in an abstract way, an Occurrence describes how that Note manifests itself in a specific container image.

As an example, a Note might be created to track the [Heartbleed](http://heartbleed.com/) vulnerability. Security vendors would then create scanners to test container images for the vulnerability, and create an Occurrence associated with each compromised container.



Along with tracking vulnerabilities, Container Analysis was designed to be a generic metadata API. Binary Authorization utilizes Container Analysis to associate signatures with the container images they are verifying\*\*.\*\* A Container Analysis Note is used to represent a single attestor, and Occurrences are created and associated with each container that attestor has approved.

The Binary Authorization API uses the concepts of "attestors" and "attestations", but these are implemented using corresponding Notes and Occurrences in the Container Analysis API.



## 6. Setting Up an Attestor

Currently, the cluster will perform a catch-all rejection on all images that don't reside at an official repository. Your next step is to create an attestor, so you can selectively allow trusted containers.

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/dca98cc118cd9139.pngAs an Attestor: |  |

## ****Creating a Container Analysis Note****

## 36f8f5ade32507f7.png

Start by creating a JSON file containing the necessary data for your Note. This command will create a JSON file containing your Note locally:

cat > ./create\_note\_request.json << EOM

{

"attestation": {

"hint": {

"human\_readable\_name": "This note represents an attestation authority"

}

}

}

EOM

Now, submit the Note to your project using the Container Analysis API:

NOTE\_ID=my-attestor-note

curl -vvv -X POST \

-H "Content-Type: application/json" \

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

--data-binary @./create\_note\_request.json \

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT\_ID}/notes/?noteId=${NOTE\_ID}"

You can verify the Note was saved by fetching it back:

curl -vvv \

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

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT\_ID}/notes/${NOTE\_ID}"

## ****Creating an Attestor in Binary Authorization****

## af0267ab7f7757f9.png

Your Note is now saved within the Container Analysis API. To make use of your attestor, you must also register the note with Binary Authorization:

ATTESTOR\_ID=my-binauthz-attestor

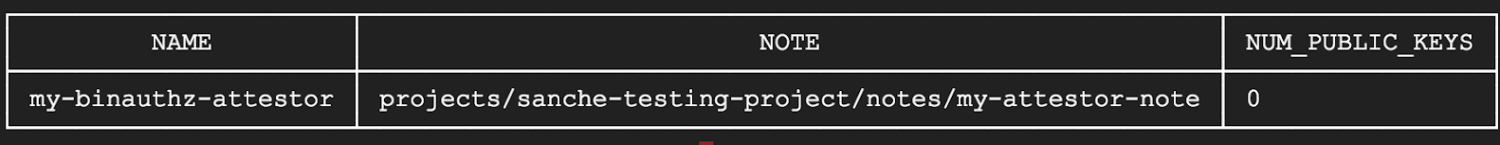
gcloud container binauthz attestors create $ATTESTOR\_ID \

--attestation-authority-note=$NOTE\_ID \

--attestation-authority-note-project=${PROJECT\_ID}

To verify everything works as expected, print out the list of registered authorities:

gcloud container binauthz attestors list



You should also be able to see your new attestor through the [Google Cloud Console UI](https://console.cloud.google.com/security/binary-authorization/attestors).

## ****Adding IAM Role****

Before you can use this attestor, you must grant Binary Authorization the appropriate permissions to view the Container Analysis Note you created. This will allow Binary Authorization to query the Container Analysis API to ensure that each pod has been signed and approved to run.

Permissions in Binary Authorization are handled through an automatically generated [service account](https://cloud.google.com/compute/docs/access/service-accounts).

First, find the service account's email address:

PROJECT\_NUMBER=$(gcloud projects describe "${PROJECT\_ID}" --format="value(projectNumber)")

BINAUTHZ\_SA\_EMAIL="service-${PROJECT\_NUMBER}@gcp-sa-binaryauthorization.iam.gserviceaccount.com"

Now, use it to create a Container Analysis IAM JSON request:

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

Make a curl request to grant the necessary IAM role:

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/${PROJECT\_ID}/notes/${NOTE\_ID}:setIamPolicy"

## ****Adding a KMS Key****

## 1e3af7c177f7a311.png

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)](https://cloud.google.com/kms/).

First, add some environment variables to describe the new key

KEY\_LOCATION=global

KEYRING=binauthz-keys

KEY\_NAME=codelab-key

KEY\_VERSION=1

Create a keyring to hold a set of keys

gcloud kms keyrings create "${KEYRING}" --location="${KEY\_LOCATION}"

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](https://pantheon.corp.google.com/security/kms) of the Google Cloud Console. Now, associate the key with your authority through the gcloud binauthz command:

gcloud beta container binauthz attestors public-keys add \

--attestor="${ATTESTOR\_ID}" \

--keyversion-project="${PROJECT\_ID}" \

--keyversion-location="${KEY\_LOCATION}" \

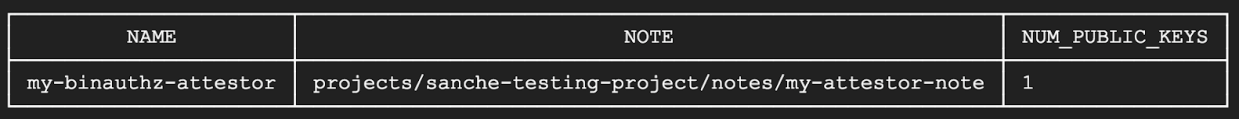
--keyversion-keyring="${KEYRING}" \

--keyversion-key="${KEY\_NAME}" \

--keyversion="${KEY\_VERSION}"

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

gcloud container binauthz attestors list



Note that multiple keys can be registered for each authority. This can be useful if the authority represents a team of people. For example, anyone in the QA team could act as the QA Attestor, and sign with their own individual private key

## 7. Signing a Container Image

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/dca98cc118cd9139.pngAs an Attestor: |  |

Now that you have your authority set up and ready to go, you can use it to sign the container image you built previously.

## ****Creating a Signed Attestation****

## 858d7e6feeb6f159.png

An attestation must include a cryptographic signature to state that a particular container image has been verified by the attestor and is safe to run on your cluster. To specify which container image to attest, you need to determine its digest. You can find the digest for a particular container tag hosted in the Container Registry by running:

DIGEST=$(gcloud container images describe ${CONTAINER\_PATH}:latest \

--format='get(image\_summary.digest)')

Now, you can use gcloud to create your attestation. The command simply 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](https://cloud.google.com/container-registry/docs/container-analysis) terms, this will create a new occurrence, and attach it to your attestor's note. To ensure everything worked as expected, you can list your attestations

gcloud container binauthz attestations list \

--attestor=$ATTESTOR\_ID --attestor-project=${PROJECT\_ID}

Now, when you attempt to run that container image, Binary Authorization will be able to determine that it was signed and verified by the attestor and it is safe to run

## 8. Running Signed Image

Now that you have your image securely verified by an attestor, let's get it running on the cluster.

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/5b1748abb8d8b699.pngAs a Policy Creator: |  |

## Updating the Policy

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

Change it to allow any images verified by the attestor:

cat << EOF > updated\_policy.yaml

globalPolicyEvaluationMode: ENABLE

defaultAdmissionRule:

evaluationMode: REQUIRE\_ATTESTATION

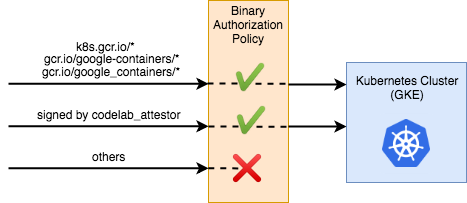
enforcementMode: ENFORCED\_BLOCK\_AND\_AUDIT\_LOG

requireAttestationsBy:

- projects/${PROJECT\_ID}/attestors/${ATTESTOR\_ID}

EOF

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.



Upload the new policy to Binary Authorization:

gcloud container binauthz policy import updated\_policy.yaml

|  |  |
| --- | --- |
| https://codelabs.developers.google.com/static/codelabs/cloud-binauthz-intro/img/4426da76922fea23.pngAs a Deployer: |  |

## Running the Verified Image

Next you'll run the signed image and verify that the pod is running with the following command:

kubectl create deployment hello-world-signed --image="${CONTAINER\_PATH}@${DIGEST}"

Now check to see if your pod is running:

kubectl get pods

You should see your pod has passed the policy and is running on the cluster.

Congratulations! You can now make specific security guarantees for your cluster by adding more complex rules to the policy.

Note: When running the container image, you must now specify the specific digest you want to deploy. Tags like "latest" will not be allowed to run on the cluster. This is because the image under a tag may change, so tags can't be secured with a signature like a digest can.

## [9. Clean Up](https://codelabs.developers.google.com/codelabs/cloud-binauthz-intro/" \l "8)

Delete the cluster:

gcloud container clusters delete binauthz-codelab --zone us-central1-a

Delete the container image:

gcloud container images delete $CONTAINER\_PATH@$DIGEST --force-delete-tags

Delete the Attestor:

gcloud container binauthz attestors delete my-binauthz-attestor

Delete the Container Analysis resources:

curl -vvv -X DELETE \

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

"https://containeranalysis.googleapis.com/v1/projects/${PROJECT\_ID}/notes/${NOTE\_ID}"