Google Kubernetes Engine Pipeline using Cloud Build

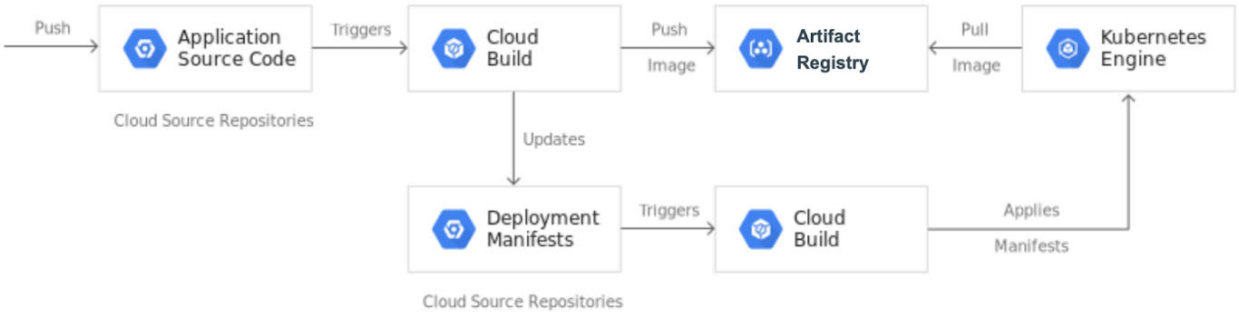
experiment Lab schedule 1 hour 30 minutes universal_currency_alt No cost

show_chart Intermediate

GSP1077

Overview

In this lab, you create a CI/CD pipeline that automatically builds a container image from committed code, stores the image in Artifact Registry, updates a Kubernetes manifest in a Git repository, and deploys the application to Google Kubernetes Engine using that manifest.



For this lab you will create 2 Git repositories:

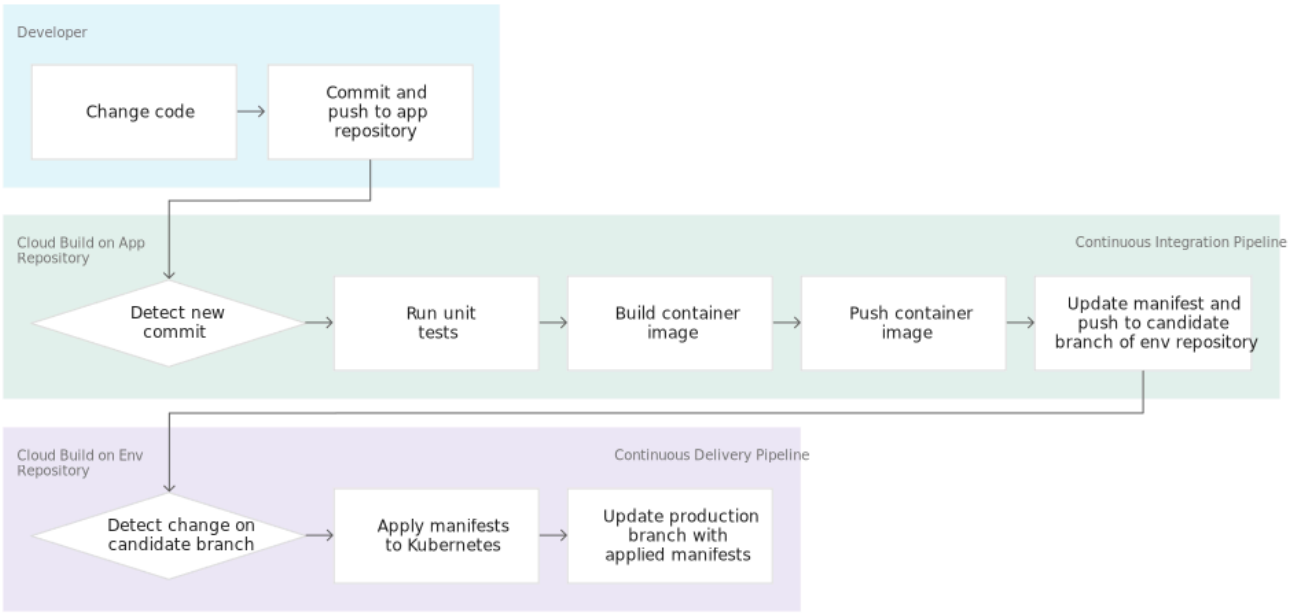
- app repository: contains the source code of the application itself
- env repository: contains the manifests for the Kubernetes Deployment

When you push a change to the **app repository**, the Cloud Build pipeline runs tests, builds a container image, and pushes it to Artifact Registry. After pushing the image, Cloud Build updates the Deployment manifest and pushes it to the **env repository**. This triggers another Cloud Build pipeline that applies the manifest to the GKE cluster and, if successful, stores the manifest in another branch of the **env repository**.

The app and env repositories are kept separate because they have different lifecycles and uses. The main users of the **app repository** are actual humans and this repository is dedicated to a specific application. The main users of the **env repository** are automated systems (such as Cloud Build), and this repository might be shared by several applications. The **env repository** can have several branches that each map to a specific environment (you only use production in this lab) and reference a specific container image, whereas the **app repository** does not.

When you finish this lab, you have a system where you can easily:

- Distinguish between failed and successful deployments by looking at the Cloud Build history.
- Access the manifest currently used by looking at the production branch of the **env repository**.
- Rollback to any previous version by re-executing the corresponding Cloud Build build.



Objectives

In this lab, you learn how to perform the following tasks:

- Create Kubernetes Engine clusters
- Create Cloud Source Repositories
- Trigger Cloud Build from Cloud Source Repositories
- Automate tests and publish a deployable container image via Cloud Build
- Manage resources deployed in a Kubernetes Engine cluster via Cloud Build

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 the **Lab Details** panel with the following:

    - The **Open Google Cloud console** button
    - Time remaining
    - The temporary credentials that you must use for this lab
    - Other information, if needed, to step through this lab

2. Click **Open Google Cloud console** (or right-click and select **Open Link in Incognito Window** if you are running the Chrome browser).

    The lab spins up resources, and then opens another tab that shows the **Sign in** page.

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

    Note: If you see the **Choose an account** dialog, click **Use Another Account**.

3. If necessary, copy the **Username** below and paste it into the **Sign in** dialog.

    ```
    student-01-097d103db006@qwiklabs.net
    ```

    content_c

    You can also find the **Username** in the **Lab Details** panel.

4. Click **Next**.

5. Copy the **Password** below and paste it into the **Welcome** dialog.

```
x3YsNF1M1KCY
```

You can also find the **Password** in the **Lab Details** panel.

6. Click **Next**.

> **Important:** You must use the credentials the lab provides you. Do not use your Google Cloud account credentials.

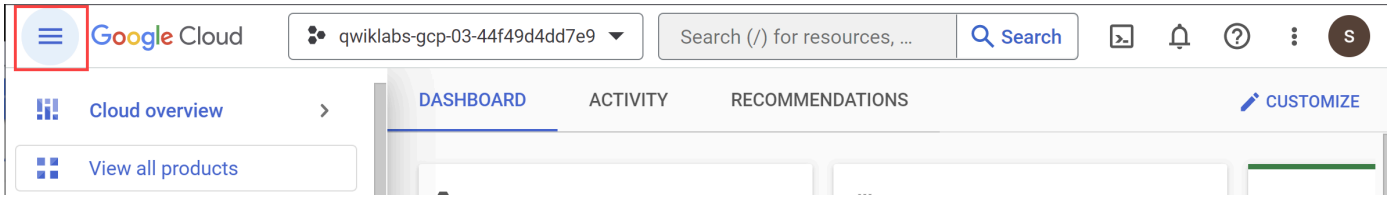
> **Note:** Using your own Google Cloud account for this lab may incur extra charges.

7. 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 Google Cloud console opens in this tab.

> **Note:** To view a menu with a list of Google Cloud products and services, click 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.

1. Click **Activate Cloud Shell** at the top of the Google Cloud console.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `qwiklabs-gcp-02-e3f86f67efb8` . The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to qwiklabs-gcp-02-e3f86f67efb
```

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

2. (Optional) You can list the active account name with this command:

```
gcloud auth list
```

content_c

3. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: student-01-097d103db006@qwiklabs.net

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

4. (Optional) You can list the project ID with this command:

```
gcloud config list project
```

content_c

Output:

```
[core]
```

```
project = qwiklabs-gcp-02-e3f86f67efb8
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Task 1. Initialize Your Lab

1. In Cloud Shell, set your project ID and project number. 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='va
export REGION=us-east4
gcloud config set compute/region $REGION
```

content_c

In the next task you will prepare your Google Cloud Project for use by enabling the required APIs, initializing the git configuration in Cloud Shell, and downloading the sample code used later in the lab.

2. Run the following command to enable the APIs for GKE, Cloud Build, Cloud Source Repositories and Container Analysis:

```
gcloud services enable container.googleapis.com \
    cloudbuild.googleapis.com \
    sourcerepo.googleapis.com \
    containeranalysis.googleapis.com
```

content_c

3. Create an Artifact Registry Docker repository named my-repository in the `us-east4` region to store your container images:

```
gcloud artifacts repositories create my-repository \
  --repository-format=docker \
```

content_c

```
--location=$REGION
```

4. Create a GKE cluster to deploy the sample application of this lab:

```
gcloud container clusters create hello-cloudbuild --num-nodes 1 --regio
```

content_c

5. If you have never used Git in Cloud Shell, configure it with your name and email address. Git will use those to identify you as the author of the commits you will create in Cloud Shell (if you don't have a github account, you can just fill in this with your current information. No account is necessary for this lab):

```
git config --global user.email "you@example.com"
```

content_c

```
git config --global user.name "Your Name"
```

content_c

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

Enable services, create an artifact registry and the GKE cluster

Check my progress

Task 2. Create the Git repositories in Cloud Source Repositories

In this task, you create the two Git repositories (**hello-cloudbuild-app** and **hello-cloudbuild-env**) and initialize **hello-cloudbuild-app** with some sample code.

1. In Cloud Shell, run the following to create the two Git repositories:

```
gcloud source repos create hello-cloudbuild-app
```

content_c

```
gcloud source repos create hello-cloudbuild-env
```

content_c

2. Clone the sample code from GitHub:

```
cd ~
```

content_c

```
git clone https://github.com/GoogleCloudPlatform/gke-gitops-tutorial-clou
```

content_c

3. Configure Cloud Source Repositories as a remote:

```
cd ~/hello-cloudbuild-app
```

content_c

```
export REGION=us-east4
sed -i "s/us-central1/$REGION/g" cloudbuild.yaml
sed -i "s/us-central1/$REGION/g" cloudbuild-delivery.yaml
sed -i "s/us-central1/$REGION/g" cloudbuild-trigger-cd.yaml
sed -i "s/us-central1/$REGION/g" kubernetes.yaml.tpl
```

content_c

```
PROJECT_ID=$(gcloud config get-value project)
```

content_c

```
git remote add google "https://source.developers.google.com/p/${PROJECT_I
```

content_c

The code you just cloned contains a simple "Hello World" application.

```
from flask import Flask
app = Flask('hello-cloudbuild')
@app.route('/')
def hello():
  return "Hello World!\n"
if __name__ == '__main__':
  app.run(host = '0.0.0.0', port = 8080)
```

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

Create the Git repositories

Check my progress

Task 3. Create a container image with Cloud Build

The code you cloned already contains the following Dockerfile.

```
FROM python:3.7-slim
RUN pip install flask
WORKDIR /app
COPY app.py /app/app.py
```

```
ENTRYPOINT ["python"]
CMD ["/app/app.py"]
```

With this Dockerfile, you can create a container image with Cloud Build and store it in Artifact Registry.

1. In Cloud Shell, create a Cloud Build build based on the latest commit with the following command:

```
cd ~/hello-cloudbuild-app
```

content_c

```
COMMIT_ID="$(git rev-parse --short=7 HEAD)"
```

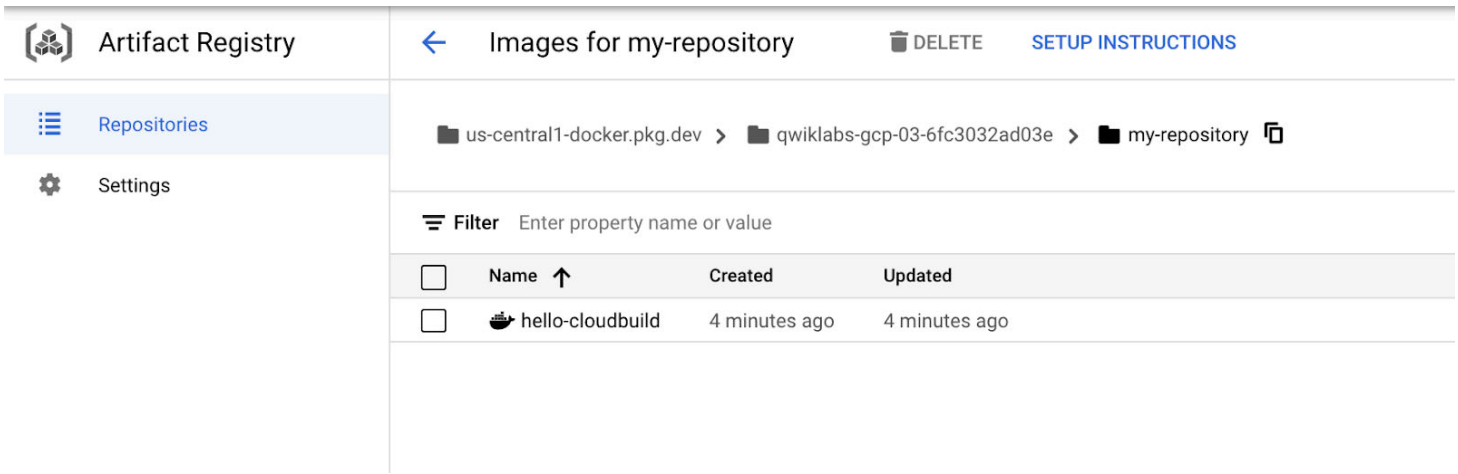
content_c

```
gcloud builds submit --tag="${REGION}-docker.pkg.dev/${PROJECT_ID}/my-rep
```

content_c

Cloud Build streams the logs generated by the creation of the container image to your terminal when you execute this command.

2. After the build finishes, in the Cloud console go to **Artifact Registry > Repositories** to verify that your new container image is indeed available in Artifact Registry. Click **my-repository**.



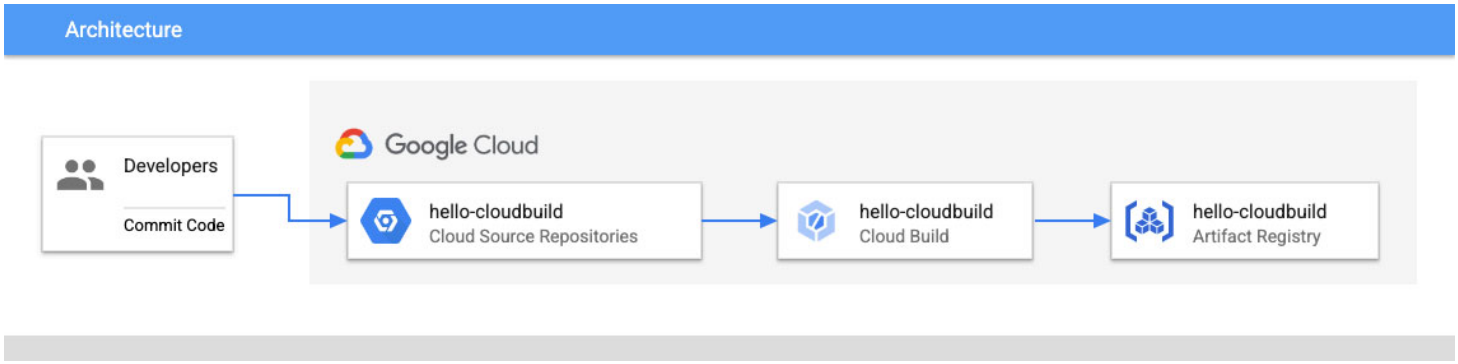
Click **Check my progress** to verify the objective.

Create the container image with Cloud Build

Check my progress

Task 4. Create the Continuous Integration (CI) pipeline

In this task, you will configure Cloud Build to automatically run a small unit test, build the container image, and then push it to Artifact Registry. Pushing a new commit to Cloud Source Repositories triggers this pipeline automatically. The **cloudbuild.yaml** file already included in the code is the pipeline's configuration.



1. In the Cloud console, go to **Cloud Build > Triggers**.
2. Click **Create Trigger**
3. In the Name field, type `hello-cloudbuild` .
4. Under **Event**, select **Push to a branch**.
5. Under **Source**, select **hello-cloudbuild-app** as your **Repository** and `.* (any branch)` as your **Branch**.
6. Under **Build configuration**, select **Cloud Build configuration file**.

7. In the **Cloud Build configuration file location** field, type `cloudbuild.yaml` after the /.

8. Click **Create**.

Source

Repository *

hello-cloudbuild-app (Cloud Source Repositories)

Select the repository to watch for events and clone when the trigger is invoked

Branch *

.*

Trigger only for a branch that matches the given regular expression Learn more

☐ Invert Regex

No branch matches

SHOW INCLUDED AND IGNORED FILES FILTERS

Configuration

Type

◉ Cloud Build configuration file (yaml or json)

○ Dockerfile

○ Buildpacks

Location

◉ Repository
hello-cloudbuild-app (Cloud Source Repositories)

○ Inline
Write inline YAML

Cloud Build configuration file location *

/ cloudbuild.yaml

Specify the path to a Cloud Build configuration file in the Git repo Learn more

When the trigger is created, return to the Cloud Shell. You now need to push the application code to Cloud Source Repositories to trigger the CI pipeline in Cloud Build.

9. To start this trigger, run the following command:

```
cd ~/hello-cloudbuild-app
```

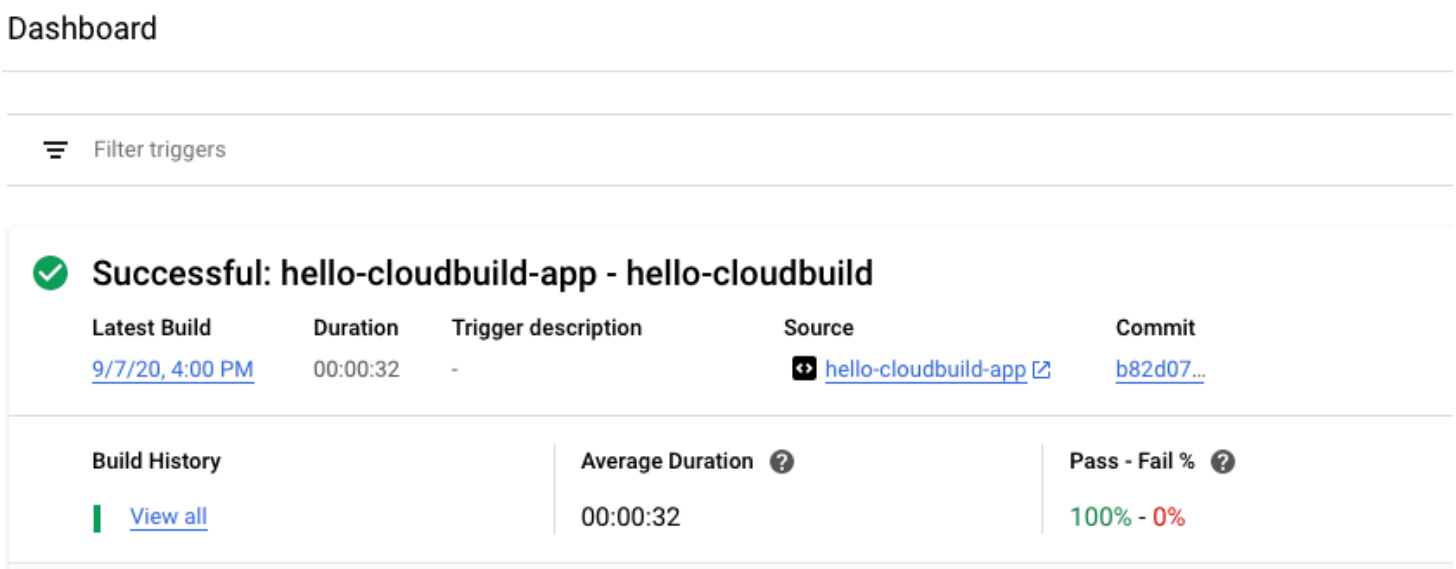
```
git add .
```

```
git commit -m "Type Any Commit Message here"
```

```
git push google master
```

10. In the Cloud console, go to **Cloud Build > Dashboard**.

11. You should see a build running or having recently finished. You can click on the build to follow its execution and examine its logs.



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

Create the Continuous Integration (CI) Pipeline

Check my progress

Task 5. Create the Test Environment and CD pipeline

Cloud Build is also used for the continuous delivery pipeline. The pipeline runs each time a commit is pushed to the candidate branch of the **hello-cloudbuild-env** repository. The pipeline applies the new version of the manifest to the Kubernetes cluster and, if successful, copies the manifest over to the production branch. This process has the following properties:

- The candidate branch is a history of the deployment attempts.
- The production branch is a history of the successful deployments.
- You have a view of successful and failed deployments in Cloud Build.
- You can rollback to any previous deployment by re-executing the corresponding build in Cloud Build. A rollback also updates the production branch to truthfully reflect the history of deployments.

Next you will modify the continuous integration pipeline to update the candidate branch of the **hello-cloudbuild-env** repository, triggering the continuous delivery pipeline.

Grant Cloud Build access to GKE

To deploy the application in your Kubernetes cluster, Cloud Build needs the Kubernetes Engine Developer Identity and Access Management role.

1. In Cloud Shell execute the following command:

```
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID} --format='get(pr
```
content_c

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

You need to initialize the **hello-cloudbuild-env** repository with two branches (production and candidate) and a Cloud Build configuration file describing the deployment process.

The first step is to clone the **hello-cloudbuild-env** repository and create the production branch. It is still empty.

2. In Cloud Shell execute the following command:

```
cd ~
```
content_c

```
gcloud source repos clone hello-cloudbuild-env
```
content_c

```
cd ~/hello-cloudbuild-env
```
content_c

```
git checkout -b production
```
content_c

3. Next you need to copy the **cloudbuild-delivery.yaml** file available in the **hello-cloudbuild-app** repository and commit the change:

```
cd ~/hello-cloudbuild-env
```
content_c

```
cp ~/hello-cloudbuild-app/cloudbuild-delivery.yaml ~/hello-cloudbuild-env
```
content_c

```
git add .
```
content_c

```
git commit -m "Create cloudbuild.yaml for deployment"
```
content_c

The `cloudbuild-delivery.yaml` file describes the deployment process to be run in Cloud Build. It has two steps:

- Cloud Build applies the manifest on the GKE cluster.
- If successful, Cloud Build copies the manifest on the production branch.

4. Create a candidate branch and push both branches for them to be available in Cloud Source Repositories:

```
git checkout -b candidate
```
content_c

```
git push origin production
```
content_c

```
git push origin candidate
```
content_c

5. Grant the Source Repository Writer IAM role to the Cloud Build service account for the **hello-cloudbuild-env** repository:

```
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID} \
--format='get(projectNumber)')"
cat >/tmp/hello-cloudbuild-env-policy.yaml <<EOF
bindings:
- members:
  - serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
  role: roles/source.writer
EOF
```
content_c

```
gcloud source repos set-iam-policy \
hello-cloudbuild-env /tmp/hello-cloudbuild-env-policy.yaml
```
content_c

Create the trigger for the continuous delivery pipeline

1. In the Cloud console, go to **Cloud Build > Triggers**.

2. Click **Create Trigger**.

3. In the Name field, type `hello-cloudbuild-deploy` .

4. Under **Event**, select **Push to a branch**.

5. Under **Source**, select **hello-cloudbuild-env** as your **Repository** and `^candidate$` as your **Branch**.

6. Under **Build configuration**, select **Cloud Build configuration file**.

7. In the **Cloud Build configuration file location** field, type `cloudbuild.yaml` after the /.

8. Click **Create**.

Source

Repository *

hello-cloudbuild-env (Cloud Source Repositories)

Select the repository to watch for events and clone when the trigger is invoked

Branch *

^candidate$

Trigger only for a branch that matches the given regular expression Learn more

☐ Invert Regex

Matches the branch: candidate

SHOW INCLUDED AND IGNORED FILES FILTERS

Configuration

Type

◉ Cloud Build configuration file (yaml or json)

○ Dockerfile

○ Buildpacks

Location

◉ Repository
hello-cloudbuild-env (Cloud Source Repositories)

○ Inline
Write inline YAML

Cloud Build configuration file location *

/ cloudbuild.yaml

Specify the path to a Cloud Build configuration file in the Git repo Learn more

Modify the continuous integration pipeline to trigger the continuous delivery pipeline.

Next, add some steps to the continuous integration pipeline that will generate a new version of the Kubernetes manifest and push it to the **hello-cloudbuild-env** repository to trigger the continuous delivery pipeline.

1. Copy the extended version of the **cloudbuild.yaml** file for the **app repository**:

```
cd ~/hello-cloudbuild-app
```

content_c

```
cp cloudbuild-trigger-cd.yaml cloudbuild.yaml
```

content_c

The **cloudbuild-trigger-cd.yaml** is an extended version of the **cloudbuild.yaml** file. It adds the steps below: they generate the new Kubernetes manifest and trigger the continuous delivery pipeline.

This pipeline uses a simple `sed` to render the manifest template. In reality, you will benefit from using a dedicated tool such as kustomize or skaffold. They allow for more control over the rendering of the manifest templates.

2. Commit the modifications and push them to Cloud Source Repositories:

```
cd ~/hello-cloudbuild-app
```

content_c

```
git add cloudbuild.yaml
```

content_c

```
git commit -m "Trigger CD pipeline"
```

content_c

```
git push google master
```

content_c

This triggers the continuous integration pipeline in Cloud Build.

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

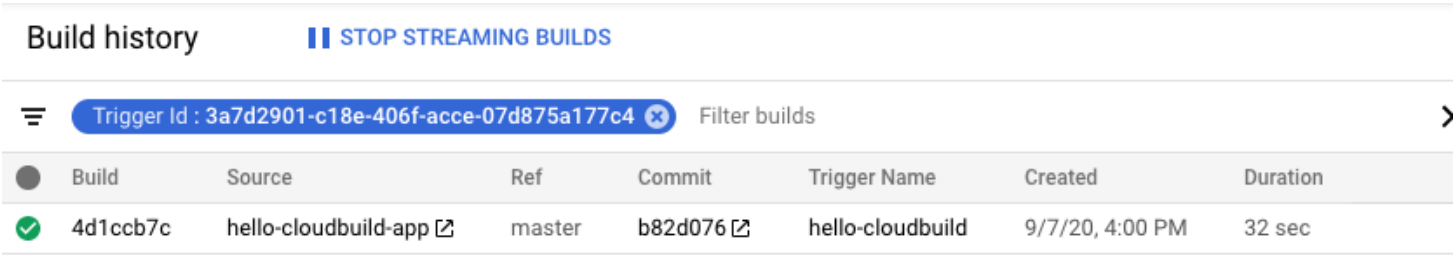
Create the Test Environment and CD Pipeline

Check my progress

Task 6. Review Cloud Build Pipeline

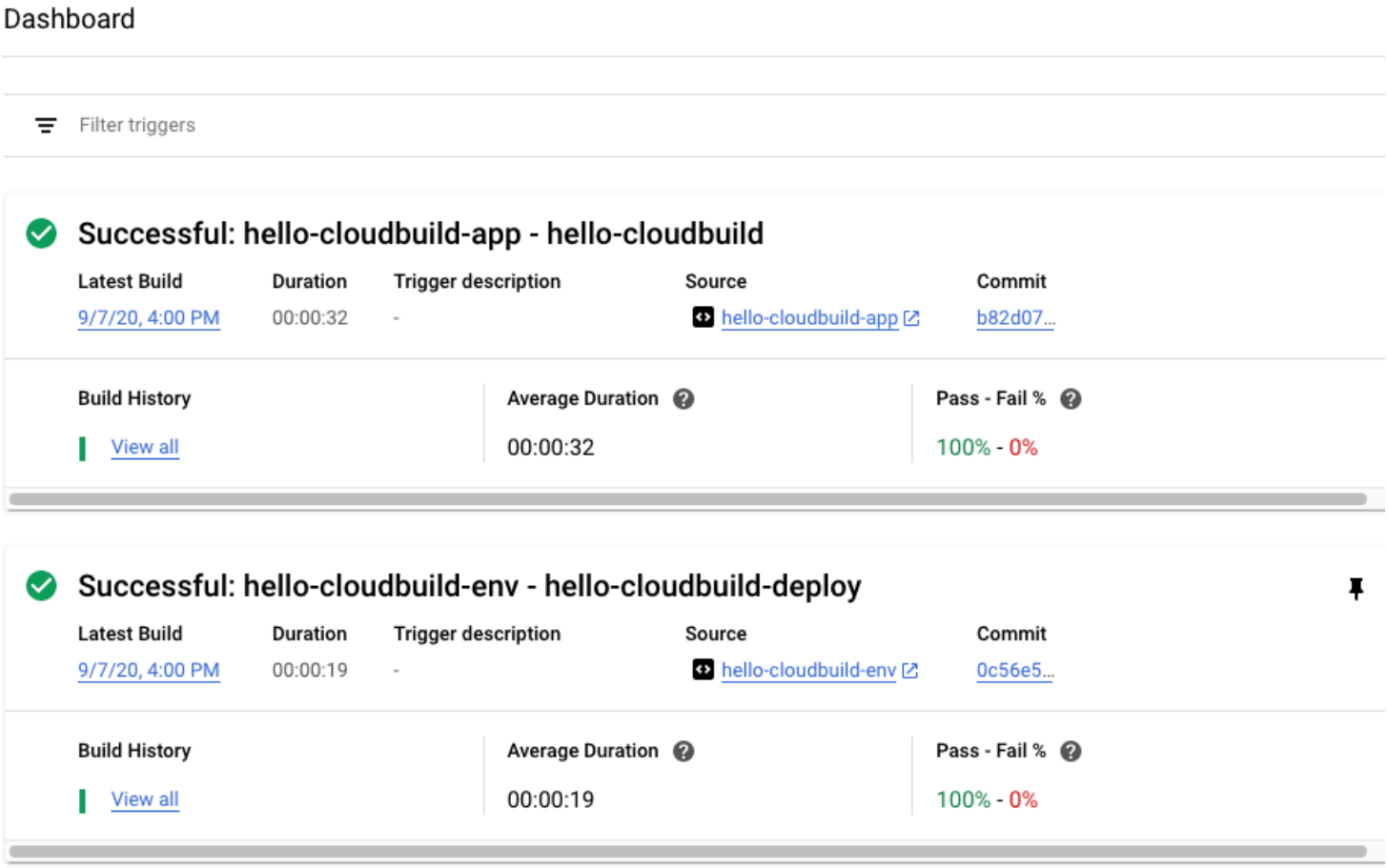
1. In the Cloud console, go to **Cloud Build > Dashboard**.

2. Click into the **hello-cloudbuild-app** trigger to follow its execution and examine its logs. The last step of this pipeline pushes the new manifest to the **hello-cloudbuild-env** repository, which triggers the continuous delivery pipeline.



3. Return to the main **Dashboard**.

4. You should see a build running or having recently finished for the **hello-cloudbuild-env** repository. You can click on the build to follow its execution and examine its logs.



Task 7. Test the complete pipeline

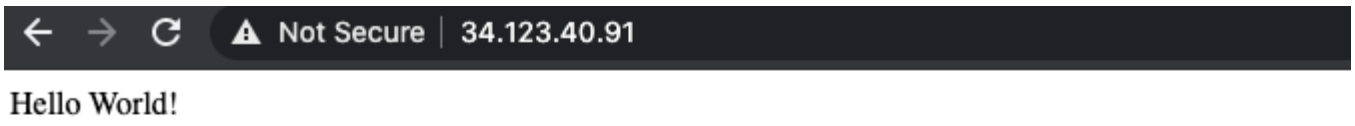
The complete CI/CD pipeline is now configured. Test it from end to end.

1. In the Cloud console, go to **Kubernetes Engine > Gateways,Services & Ingress**.

There should be a single service called **hello-cloudbuild** in the list. It has been created by the continuous delivery build that just ran.

2. Click on the endpoint for the **hello-cloudbuild** service. You should see "Hello World!". If there is no endpoint, or if you see a load balancer error, you may have to wait a few minutes for the load balancer to

be completely initialized. Click **Refresh** to update the page if needed.



3. In Cloud Shell, replace "Hello World" with "Hello Cloud Build", both in the application and in the unit test:

```
cd ~/hello-cloudbuild-app
```

content_c

```
sed -i 's/Hello World/Hello Cloud Build/g' app.py
```

content_c

```
sed -i 's/Hello World/Hello Cloud Build/g' test_app.py
```

content_c

4. Commit and push the change to Cloud Source Repositories:

```
git add app.py test_app.py
```

content_c

```
git commit -m "Hello Cloud Build"
```

content_c

```
git push google master
```

content_c

5. This triggers the full CI/CD pipeline.

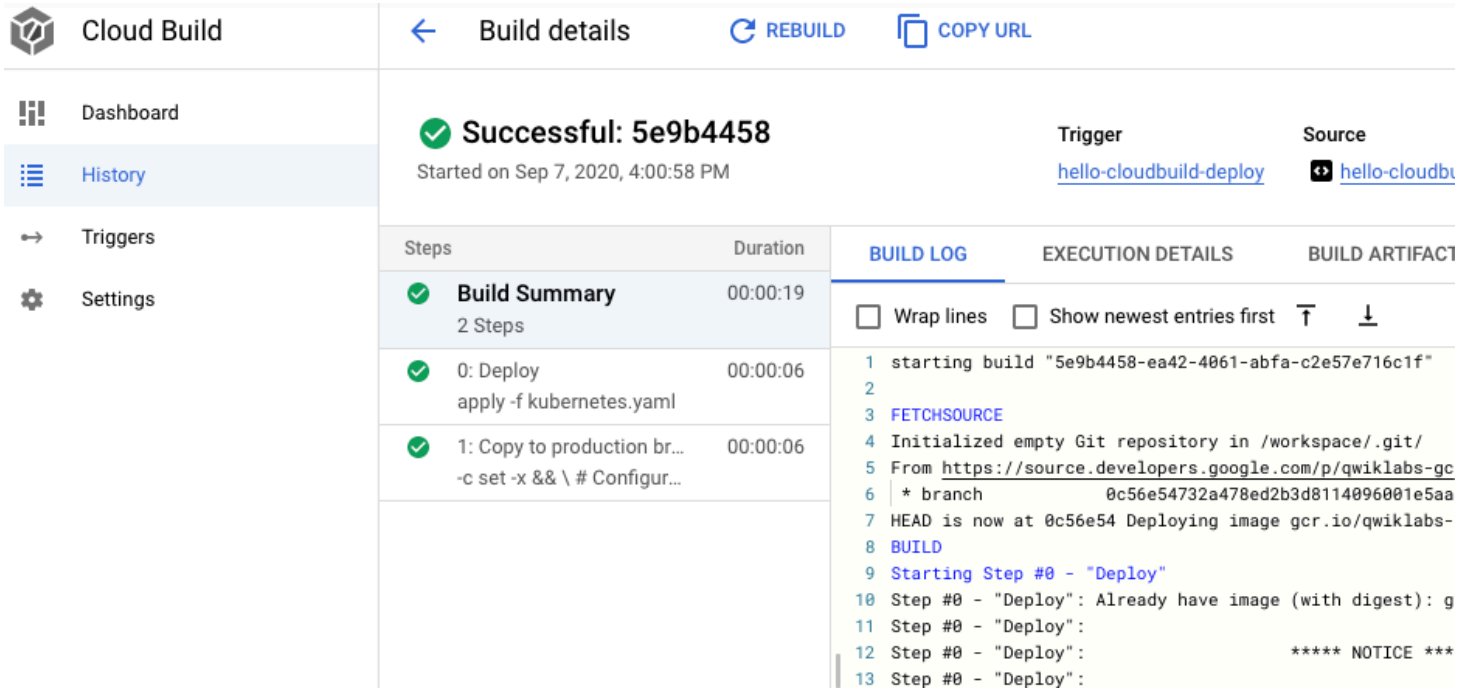
After a few minutes, reload the application in your browser. You should now see "Hello Cloud Build!".

Hello Cloud Build!

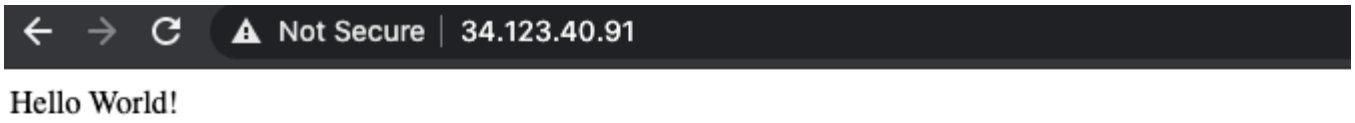
Task 8. Test the rollback

In this task, you rollback to the version of the application that said "Hello World!".

1. In the Cloud console, go to **Cloud Build > Dashboard**.
2. Click on *View all* link under **Build History** for the **hello-cloudbuild-env** repository.
3. Click on the second most recent build available.
4. Click **Rebuild**.



When the build is finished, reload the application in your browser. You should now see "Hello World!" again.



Hello World!

Congratulations!

Now you can use Cloud Build to create and rollback continuous integration pipelines with GKE on Google Cloud!

Google Cloud training and 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: January 26, 2024

Lab Last Tested: January 19, 2024