


Google Kubernetes Engine Pipeline using Cloud Build

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.
2. Click through the following windows:
 - Continue through the Cloud Shell information window.
 - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `qwiklabs-gcp-01-99125b73e6bb`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to quiklabs-gcp-01-99125b73e6bb
```

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

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

```
gcloud auth list
```

Copied!

content_copy

4. Click **Authorize**.

Output:

```
ACTIVE: *  
ACCOUNT: student-02-dbdd26f73642@quiklabs.net
```

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

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

```
gcloud config list project
Copied!
```

content_copy

Output:

```
[core]
project = qwiklabs-gcp-01-99125b73e6bb
```

Note: For full documentation of gcloud, in Google Cloud, refer to [the gcloud CLI overview guide](#).

Task 1. Initialize your lab

In this task, you set up your environment:

- Import your project ID and project number as variables
- Enable the APIs for GKE, Cloud Build, Secret Manager, and Artifact Analysis
- Create an Artifact Registry Docker repository
- Create a GKE cluster to deploy the sample application of this lab
 1. In Cloud Shell, run the following command to 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='value(projectNumber)')
export REGION=us-east1
gcloud config set compute/region $REGION
Copied!
```

content_copy

Next, you 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, Secret Manager and Artifact Analysis:

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

Copied!

content_copy

3. Create an Artifact Registry Docker repository named `my-repository` in the `us-east1` region to store your container images:

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

Copied!

content_copy

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

```
gcloud container clusters create hello-cloudbuild --num-nodes 1 --
region $REGION
```

Copied!

content_copy

5. Run the following command to configure Git and GitHub in Cloud Shell:

```
6. curl -sS https://webi.sh/gh | sh
7. gh auth login
8. gh api user -q ".login"
9. GITHUB_USERNAME=$(gh api user -q ".login")
10. git config --global user.name "${GITHUB_USERNAME}"
11. git config --global user.email "${USER_EMAIL}"
12. echo ${GITHUB_USERNAME}
    echo ${USER_EMAIL}
```

Copied!

content_copy

Press ENTER to accept the default options. Read the instructions in the CLI tool to GitHub Login with a web browser. If you have logged in successfully, it shows your GitHub username.

Task 2. Create the Git repositories in GitHub repositories

GitHub is a platform where you can store, share, and work together with others to write code. Git is a version control system. When you upload files to GitHub, you

store them in a "Git repository." This means that when you make changes (or "commits") to your files in GitHub, Git automatically starts to track and manage your changes. For more detail, refer to [About GitHub and Git](#).

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 commands to create the two Git repositories:

```
gh repo create hello-cloudbuild-app --private
```

Copied!

content_copy

```
gh repo create hello-cloudbuild-env --private
```

Copied!

content_copy

2. Download the sample code from Cloud Storage:

```
cd ~
mkdir hello-cloudbuild-app
```

Copied!

content_copy

```
gcloud storage cp -r gs://spls/gsp1077/gke-gitops-tutorial-cloudbuild/*
hello-cloudbuild-app
```

Copied!

content_copy

3. Configure the GitHub repository as a remote:

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

Copied!

content_copy

```
export REGION=us-east1
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
```

Copied!

content_copy

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

Copied!

content_copy

```
git init
git config credential.helper gcloud.sh
git remote add google https://github.com/${GITHUB_USERNAME}/hello-
cloudbuild-app
git branch -m master
git add . && git commit -m "initial commit"
```

Copied!

content_copy

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)
```

Task 3. Create a container image with Cloud Build

In this task, with an existing Dockerfile, you use Cloud Build to create and store a container image.

The code you previously cloned contains the Docker file:

```
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 use Cloud Build to create a container image 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
```

Copied!

content_copy

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

Copied!

content_copy

```
gcloud builds submit --tag="${REGION}-docker.pkg.dev/${PROJECT_ID}/my-repository/hello-cloudbuild:${COMMIT_ID}" .
```

Copied!

content_copy

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

2. After the build finishes, in the Google title bar, enter **Artifact Registry** in the **Search** field, and then click **Artifact Registry** in the search results. Verify that your new container image is indeed available in Artifact Registry. Click **my-repository** to see the `hello-cloudbuild` image in the **Image** list.

Task 4. Create the Continuous Integration (CI) pipeline

In this task, you 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 GitHub repositories triggers this pipeline automatically.

The `cloudbuild.yaml` file, already included in the code, is the pipeline's configuration.

1. In the console title bar, enter **Cloud Build triggers** in the **Search** field, and then click **Triggers, Cloud Build** in the search results.
2. Click **Create Trigger**.
3. For **Name**, type `hello-cloudbuild`. Set **Region** to `us-east1`.
4. Set **Event** to **Push to a branch**.
5. Under **Source**, for **Repository**, click **Connect new repository**.

a. Select **GitHub (Cloud Build GitHub App)**. Click **Continue**.

b. Authenticate to your source repository with your username and password.

c. If you get the pop up "The GitHub App is not installed on any of your repositories", follow these steps.

i. Click **Install Google Cloud Build**. Install the Cloud Build GitHub App in your personal account. Permit the installation using your GitHub account.

ii. Under **Repository access**. Choose **Only select repositories**. Click the **Select the repositories** menu and select `\${GITHUB_USERNAME}/hello-cloudbuild-app` and `\${GITHUB_USERNAME}/hello-cloudbuild-env`.

iii. Click **Install**.

Copied!

content_copy

d. Select `\${GITHUB_USERNAME}/hello-cloudbuild-app` for **Repository**. Click **OK**.

e. Accept **I understand that GitHub content for the selected repositories....**

f. Click **Connect**.

6. If the Cloud Build GitHub App is already installed in your account, you get the option to **Edit Repositories** on GitHub.

a. Under **Repository access** choose **Only select repositories**. Click the **Select repositories** menu and select the repository

`\${GITHUB_USERNAME}/hello-cloudbuild-app` and

`\${GITHUB_USERNAME}/hello-cloudbuild-env`.

b. Click **Save**.

7. On the Trigger page, from the **Repository list**, click `\${GITHUB_USERNAME}/hello-cloudbuild-app`.

8. For **Branch** type **.*** (any branch).

9. In the **Configuration** section, set **Type** to **Cloud Build configuration file**.

10. In the **Location** field, type `cloudbuild.yaml` after the `/`.

11. Set **Service account** to the **Compute Engine default service account**.

12. Click **Create**.

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

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

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

Copied!

```
content_copy
```

```
git add .
```

Copied!

```
content_copy
```

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

Copied!

```
content_copy
```

```
git push google master
```

Copied!

```
content_copy
```

14. In the left pane, click **Dashboard**.

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

Task 5. Accessing GitHub from a build via SSH keys

In this step use the Secret Manager with Cloud Build to access private GitHub repositories.

Create a SSH key

1. In Cloud Shell, change to the home directory.


```
cd ~
```

Copied!

```
content_copy
```

2. Create a new directory named `workingdir` and navigate to it:

3.

```
mkdir workingdir
```

```
cd workingdir
```

Copied!

```
content_copy
```

4. Create a new GitHub SSH key, replace `[your-github-email]` with your personal GitHub email address:

```
ssh-keygen -t rsa -b 4096 -N '' -f id_github -C [your-github-email]
```

Copied!

```
content_copy
```

This step creates two files, `id_github` and `id_github.pub`.

4. In the Cloud Shell action bar, click **More (⋮)** and then **Download > Toggle file browser** and select the dropdown and `workingdir` folder to download the `id_github` file on your local machine.

Store the private SSH key in Secret Manager

1. In the console title bar, enter **Secret Manager**, and then click **Secret Manager** in the search results.
2. Click **Create Secret**.
3. Set **Name** to `ssh_key_secret`.
4. Set **Secret value** to **Upload** and upload your `id_github` file.
5. Leave other settings at their defaults.

6. Click **Create secret**.

This uploads your `id_github` file to Secret Manager.

Add the public SSH key to your private repository's deploy keys

1. Login to your personal [GitHub](#) account
2. In the top right corner, click your profile photo, then click **Your profile**.
3. On your profile page, click **Repositories**, then click the `hello-cloudbuild-env` repository.
4. From your repository, click **Settings**.
5. In the left pane, click **Deploy Keys**, then click **Add deploy key**.
6. Provide the title **SSH_KEY**, paste your public SSH key from `workingdir/id_github.pub` from Cloud Shell.
7. Select **Allow write access** so this key has write access to the repository. A deploy key with write access lets a deployment push to the repository.
8. Click **Add key**.
9. Delete the SSH key from your disk:

```
rm id_github*  
Copied!  
content_copy
```

Grant the service account permission to access Secret Manager

Enter the following command to give the service account access to Secret Manager:

```
gcloud projects add-iam-policy-binding ${PROJECT_NUMBER} \
```

```
--member=serviceAccount:${PROJECT_NUMBER}-  
compute@developer.gserviceaccount.com \  
--role=roles/secretmanager.secretAccessor
```

Task 6. Create the test environment and CD pipeline

You can also use Cloud Build 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 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 the Access Management role.

1. In Cloud Shell, execute the following command:

```
cd ~  
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID} --  
format='get(projectNumber) ')"
```

Copied!

content_copy

```
gcloud projects add-iam-policy-binding ${PROJECT_NUMBER} \  

```

```
--
member=serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
\
--role=roles/container.developer
Copied!
```

content_copy

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, which is still empty.

2. In Cloud Shell, download the sample code from Cloud Storage:

```
mkdir hello-cloudbuild-env
gcloud storage cp -r gs://spls/gsp1077/gke-gitops-tutorial-cloudbuild/*
hello-cloudbuild-env
Copied!
```

content_copy

```
cd hello-cloudbuild-env
export REGION=us-east1
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
Copied!
```

content_copy

3. For Cloud Build to connect to GitHub, you must add the public SSH key to the `known_hosts` file in Cloud Build's build environment. In your `hello-cloudbuild-env` directory, create a file named `known_hosts.github`, add the public SSH key to this file, and provide the necessary permission to the file:

```
cd ~/hello-cloudbuild-env
ssh-keyscan -t rsa github.com > known_hosts.github
chmod +x known_hosts.github
Copied!
```

content_copy

```
git init
git config credential.helper gcloud.sh
git remote add google https://github.com/${GITHUB_USERNAME}/hello-
cloudbuild-env
git branch -m master
git add . && git commit -m "initial commit"
git push google master
Copied!
```

content_copy

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

Copied!

content_copy

```
git checkout -b production
```

Copied!

content_copy

4. Next, replace the `cloudbuild.yaml` file available in the `hello-cloudbuild-env` repository and commit the change:

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

Copied!

content_copy

5. Replace the `cloudbuild.yaml` in the `hello-cloudbuild-env` repository with the code below. Replace `{GITHUB-USERNAME}` with your personal GitHub username:

```
# Copyright 2018 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# [START cloudbuild-delivery]
steps:
# This step deploys the new version of our container image
# in the hello-cloudbuild Kubernetes Engine cluster.
- name: 'gcr.io/cloud-builders/kubect1'
  id: Deploy
  args:
  - 'apply'
  - '-f'
  - 'kubernetes.yaml'
  env:
  - 'CLOUDSDK_COMPUTE_REGION=us-east1'
  - 'CLOUDSDK_CONTAINER_CLUSTER=hello-cloudbuild'

# Access the id_github file from Secret Manager, and setup SSH
- name: 'gcr.io/cloud-builders/git'
  secretEnv: ['SSH_KEY']
  entrypoint: 'bash'
  args:
  - -c
  - |
    echo "$SSH_KEY" >> /root/.ssh/id_rsa
    chmod 400 /root/.ssh/id_rsa
    cp known_hosts.github /root/.ssh/known_hosts
```

```

volumes:
- name: 'ssh'
  path: /root/.ssh

# Clone the repository
- name: 'gcr.io/cloud-builders/git'
  args:
  - clone
  - --recurse-submodules
  - git@github.com:${GITHUB-USERNAME}/hello-cloudbuild-env.git
  volumes:
  - name: ssh
    path: /root/.ssh

# This step copies the applied manifest to the production branch
# The COMMIT_SHA variable is automatically
# replaced by Cloud Build.
- name: 'gcr.io/cloud-builders/gcloud'
  id: Copy to production branch
  entrypoint: /bin/sh
  args:
  - '-c'
  - |
    set -x && \
    cd hello-cloudbuild-env && \
    git config user.email $(gcloud auth list --filter=status:ACTIVE --
format='value(account)')
    sed "s/GOOGLE_CLOUD_PROJECT/${PROJECT_ID}/g" kubernetes.yaml.tpl |
\
    git fetch origin production && \
    # Switch to the production branch and copy the kubernetes.yaml file
from the candidate branch
    git checkout production && \
    git checkout $COMMIT_SHA kubernetes.yaml && \
    # Commit the kubernetes.yaml file with a descriptive commit message
    git commit -m "Manifest from commit $COMMIT_SHA
$(git log --format=%B -n 1 $COMMIT_SHA)" && \
    # Push the changes back to Cloud Source Repository
    git push origin production
  volumes:
  - name: ssh
    path: /root/.ssh

availableSecrets:
  secretManager:
  - versionName:
projects/${PROJECT_NUMBER}/secrets/ssh_key_secret/versions/1
  env: 'SSH_KEY'

# [END cloudbuild-delivery]
options:
  logging: CLOUD_LOGGING_ONLY
Copied!

content_copy

git add .
Copied!

```

content_copy

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

Copied!

content_copy

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

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

6. Create a candidate branch and push both branches for them to be available in GitHub Repositories:

```
git checkout -b candidate
```

Copied!

content_copy

```
git push google production
```

Copied!

content_copy

```
git push google candidate
```

Copied!

content_copy

Create the trigger for the continuous delivery pipeline

1. In the console title bar, enter **Cloud Build Triggers**, and then click **Triggers, Cloud Build**.
2. Click **Create Trigger**.
3. Set **Name** to **hello-cloudbuild-deploy**. Set **Region** to `us-east1`.
4. Under **Event**, select **Push to a branch**.
5. Under **Source**, for **Repository** click **Connect new repository**.
 - a. Select **GitHub (Cloud Build GitHub App)**. Click **Continue**.
 - b. Authenticate to your source repository with your GitHub username and password.

- c. Select `${GITHUB_USERNAME}/hello-cloudbuild-env` repository.
Click **OK**.
 - d. Select **I understand that GitHub content for the selected repositories..**
 - e. Click **Connect**.
6. Under **Repository** select `${GITHUB_USERNAME}/hello-cloudbuild-env`.
 7. Under **Source**, select `^candidate$` as your **Branch**.
 8. Under **Build configuration**, select **Cloud Build configuration file**.
 9. In the **Cloud Build configuration file location** field,
type `cloudbuild.yaml` after the `/`.
 10. Set **Service account** to the Compute Engine default service account.
 11. Click **Create**.
 12. In your `hello-cloudbuild-app` directory, create a file
named `known_hosts.github`, add the public SSH key to this file and provide
the necessary permission to the file:

```
cd ~/hello-cloudbuild-app
ssh-keyscan -t rsa github.com > known_hosts.github
chmod +x known_hosts.github
Copied!
```

content_copy

```
git add .
git commit -m "Adding known_host file."
git push google master
Copied!
```

content_copy

Modify the continuous integration pipeline to trigger the continuous delivery pipeline

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

- Copy the extended version of the `cloudbuild.yaml` file for the app repository:

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

Copied!

content_copy

The `cloudbuild.yaml` file adds the steps that generate the new Kubernetes manifest and trigger the continuous delivery pipeline.

Configure the build

1. Replace the `cloudbuild.yaml` in the `hello-cloudbuild-app` repository with the code below. Replace `${GITHUB-USERNAME}` with your GitHub username.

```
# Copyright 2018 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# [START cloudbuild]
steps:
# This step runs the unit tests on the app
- name: 'python:3.7-slim'
  id: Test
  entrypoint: /bin/sh
  args:
    - -c
    - 'pip install flask && python test_app.py -v'

# This step builds the container image.
- name: 'gcr.io/cloud-builders/docker'
  id: Build
  args:
    - 'build'
    - '-t'
    - 'us-east1-docker.pkg.dev/$PROJECT_ID/my-repository/hello-
cloudbuild:$SHORT_SHA'
    - '.'

# This step pushes the image to Artifact Registry
# The PROJECT_ID and SHORT_SHA variables are automatically
```

```

# replaced by Cloud Build.
- name: 'gcr.io/cloud-builders/docker'
  id: Push
  args:
    - 'push'
    - 'us-east1-docker.pkg.dev/${PROJECT_ID}/my-repository/hello-
cloudbuild:${SHORT_SHA}'
# [END cloudbuild]

# Access the id_github file from Secret Manager, and setup SSH
- name: 'gcr.io/cloud-builders/git'
  secretEnv: ['SSH_KEY']
  entrypoint: 'bash'
  args:
    - -c
    - |
      echo "$SSH_KEY" >> /root/.ssh/id_rsa
      chmod 400 /root/.ssh/id_rsa
      cp known_hosts.github /root/.ssh/known_hosts
  volumes:
    - name: 'ssh'
      path: /root/.ssh

# Clone the repository
- name: 'gcr.io/cloud-builders/git'
  args:
    - clone
    - --recurse-submodules
    - git@github.com:${GITHUB_USERNAME}/hello-cloudbuild-env.git
  volumes:
    - name: ssh
      path: /root/.ssh

# [START cloudbuild-trigger-cd]
# This step clones the hello-cloudbuild-env repository
- name: 'gcr.io/cloud-builders/gcloud'
  id: Change directory
  entrypoint: /bin/sh
  args:
    - '-c'
    - |
      cd hello-cloudbuild-env && \
      git checkout candidate && \
      git config user.email $(gcloud auth list --filter=status:ACTIVE --
format='value(account)')
  volumes:
    - name: ssh
      path: /root/.ssh

# This step generates the new manifest
- name: 'gcr.io/cloud-builders/gcloud'
  id: Generate manifest
  entrypoint: /bin/sh
  args:
    - '-c'
    - |
      sed "s/GOOGLE_CLOUD_PROJECT/${PROJECT_ID}/g" kubernetes.yaml.tpl |
\

```

```

    sed "s/COMMIT_SHA/${SHORT_SHA}/g" > hello-cloudbuild-
env/kubernetes.yaml
  volumes:
  - name: ssh
    path: /root/.ssh

# This step pushes the manifest back to hello-cloudbuild-env
- name: 'gcr.io/cloud-builders/gcloud'
  id: Push manifest
  entrypoint: /bin/sh
  args:
  - '-c'
  - |
    set -x && \
    cd hello-cloudbuild-env && \
    git add kubernetes.yaml && \
    git commit -m "Deploying image us-east1-
docker.pkg.dev/$PROJECT_ID/my-repository/hello-cloudbuild:${SHORT_SHA}
Built from commit ${COMMIT_SHA} of repository hello-cloudbuild-app
Author: $(git log --format='%an <ae>' -n 1 HEAD)" && \
    git push origin candidate
  volumes:
  - name: ssh
    path: /root/.ssh
availableSecrets:
  secretManager:
  - versionName:
projects/${PROJECT_NUMBER}/secrets/ssh_key_secret/versions/1
  env: 'SSH_KEY'

# [END cloudbuild-trigger-cd]
options:
  logging: CLOUD_LOGGING_ONLY
Copied!

```

content_copy

Note: This pipeline uses a simple `sed` to render the manifest template. In reality, you benefit from using a dedicated tool such as `kustomize` or `scaffold` as they allow more control over manifest template rendering.

2. Commit the modifications and push them to GitHub Repositories:

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

Copied!

content_copy

```
git add cloudbuild.yaml
```

Copied!

content_copy

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

Copied!

content_copy

```
git push google master
```

Copied!

content_copy

This triggers the continuous integration pipeline in Cloud Build.

Task 7. Review Cloud Build pipeline

In this task, you review the Cloud Build pipeline in the console.

1. In the console, still in the Cloud Build page, click **Dashboard** in the left pane.
2. Click 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 either running or recently finished for the `hello-cloudbuild-env` repository.

You can click on the build to follow its execution and examine its logs.

Task 8. Test the complete pipeline

You've now configured the complete CI/CD pipeline. In this task you perform an end to end test.

1. In the console, in the **Navigation menu** () , click **Kubernetes Engine > Gateways, Services & Ingress > Services**.

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 completely initialize. If needed, click **Refresh** to update the page.

Hello World!

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

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

Copied!

content_copy

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

Copied!

content_copy

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

Copied!

content_copy

4. Commit and push the change to GitHub repositories:

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

Copied!

content_copy

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

Copied!

content_copy

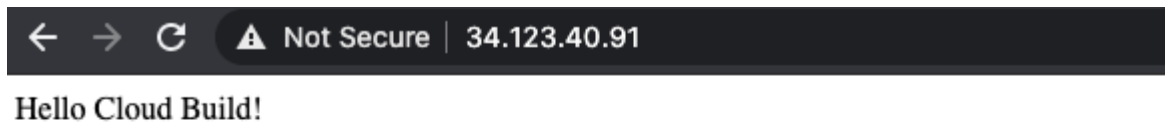
```
git push google master
```

Copied!

content_copy

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!".



Task 9. Test the rollback

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

1. In the console title bar, type **Cloud Build Dashboard** in the **Search** field, and then click **Cloud Build** in the search results. Be sure **Dashboard** is selected in the left pane.
2. Click the **View all** link under **Build History** for the `hello-cloudbuild-env` repository.
3. Click on the second most recent build available.
4. Click **Rebuild**.

A screenshot of the Google Cloud Build Dashboard. The left sidebar shows the navigation menu with 'Dashboard', 'History', 'Triggers', and 'Settings'. The main area shows 'Build details' for a successful build with ID '5e9b4458'. It includes a table of steps, a build log, and execution details.

Steps	Duration	BUILD LOG	EXECUTION DETAILS	BUILD ARTIFACTS
Build Summary 2 Steps	00:00:19	<input type="checkbox"/> Wrap lines <input type="checkbox"/> Show newest entries first		
0: Deploy apply -f kubernetes.yaml	00:00:06	1 starting build "5e9b4458-ea42-4061-abfa-c2e57e716c1f"		
1: Copy to production br... -c set -x && \ # Configur...	00:00:06	2 3 FETCHSOURCE 4 Initialized empty Git repository in /workspace/.git/ 5 From https://source.developers.google.com/p/qwiklabs-gc 6 * branch 7 HEAD is now at 0c56e54 Deploying image gcr.io/qwiklabs- 8 BUILD 9 Starting Step #0 - "Deploy" 10 Step #0 - "Deploy": Already have image (with digest): g 11 Step #0 - "Deploy": 12 Step #0 - "Deploy": 13 Step #0 - "Deploy":		

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

← → ↻ ⚠ Not Secure | 34.123.40.91

Hello World!