Continuous Deployment with Spinnaker v1.6

3 hours9 Credits

Rate Lab

This lab shows you how to create a continuous delivery pipeline using Google Kubernetes Engine, Cloud Source Repositories, Cloud Build, and Spinnaker. After you create a sample app, you configure these services to automatically build, test, and deploy it. When you modify the app code, the changes trigger the continuous delivery pipeline to automatically rebuild, retest, and redeploy the new version.

Objectives

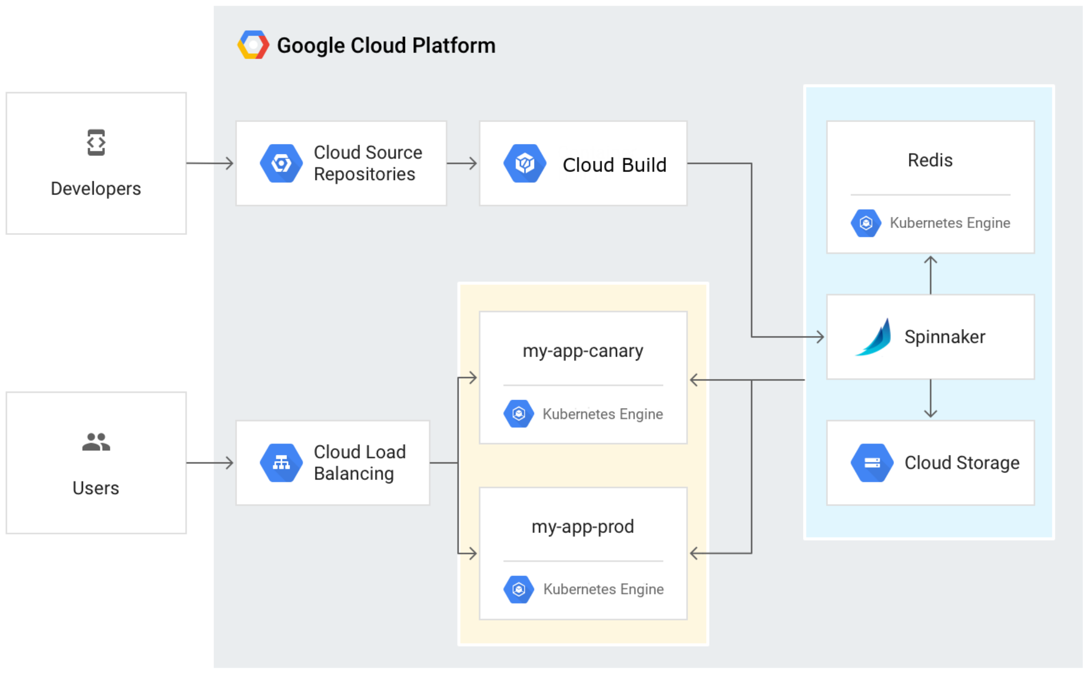
- Set up your environment by launching Cloud Shell, creating a GKE cluster, and configuring your identity and user management scheme.
- Download a sample app, create a Git repository, and upload it to a Cloud Source Repository.

- Deploy Spinnaker to GKE using Helm.
- Build your Docker image.

- Create triggers to rebuild Docker images when your app changes.

- Configure a Spinnaker pipeline to reliably and continuously deploy your app to GKE.
- Deploy a code change, triggering the pipeline, and watch it roll out to production.

Pipeline architecture



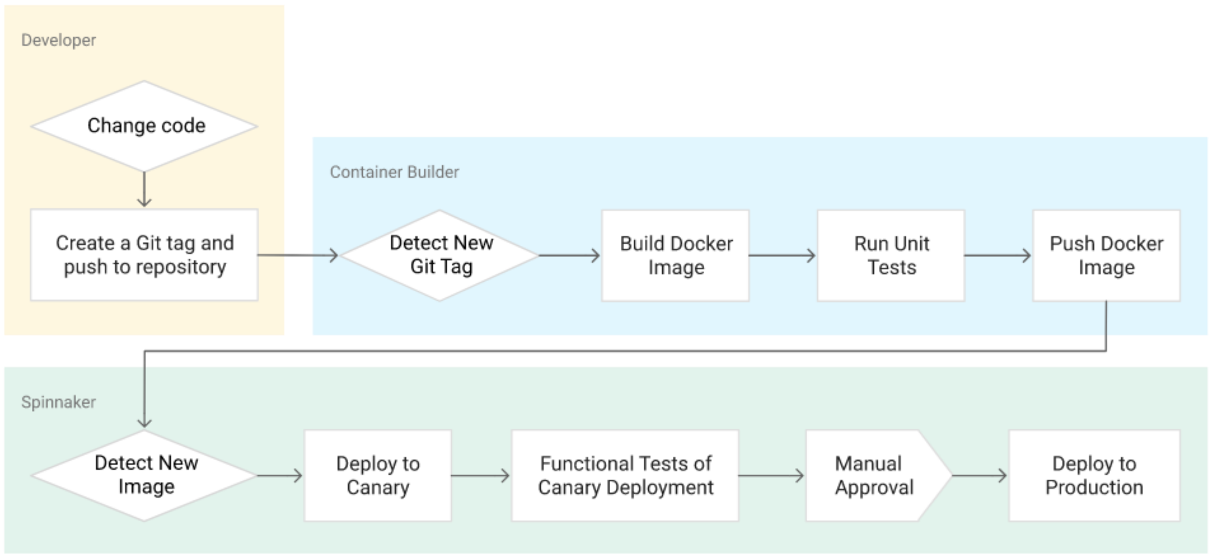
To continuously deliver app updates to your users, you need an automated process that reliably builds, tests, and updates your software. Code changes should automatically flow through a pipeline that includes artifact creation, unit testing, functional testing, and production rollout. In some cases, you want a code update to apply to only a subset of your users, so that it is exercised realistically before you push it to your entire user base. If one of these canary releases proves

unsatisfactory, your automated procedure must be able to quickly roll back the software changes.
With GKE and Spinnaker, you can create a robust continuous delivery flow that helps to ensure your software is shipped as quickly as it is developed and validated. Although rapid iteration is your end goal, you must first ensure that each app revision passes through a series of automated validations before becoming a candidate for production rollout. When a given change has been vetted through automation, you can also validate the app manually and conduct further prerelease testing.

After your team decides the app is ready for production, one of your team members can approve it for production deployment.

App delivery pipeline

In this lab, you build the continuous delivery pipeline shown in the following diagram.



Setup and requirements

Qwiklabs setup

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click Start Lab, shows how long Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access the Google Cloud Platform for the duration of the lab.

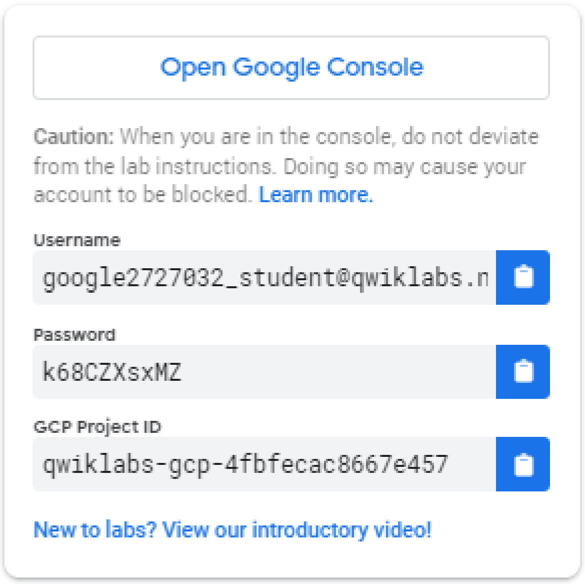
What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal GCP account or project, do not use it for this lab.

How to start your lab and sign in to the Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username

google2727032_student@qwiklabs.n

Password

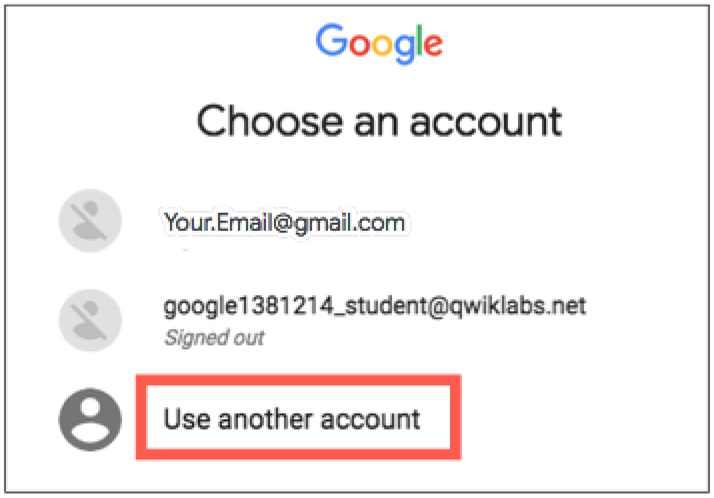
k68CZXsxMZ

GCP Project ID

qwiklabs-gcp-4fbfecac8667e457

New to labs? View our introductory video!

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.
   Tip: Open the tabs in separate windows, side-by-side.
3. On the Choose an account page, click **Use Another Account**.



Your.Email@gmail.com

google1381214_student@qwiklabs.net
Signed out

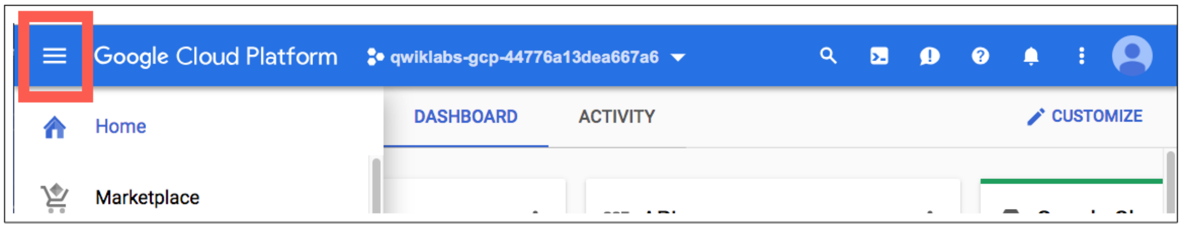
4. The Sign in page opens. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

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

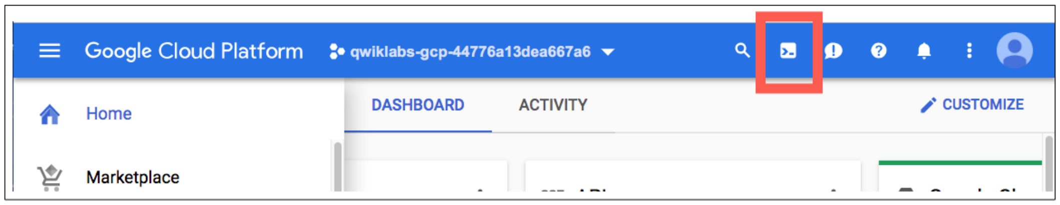
Note: You can view the menu with a list of GCP Products and Services by clicking the **Navigation menu** at the top-left, next to “Google Cloud Platform”.



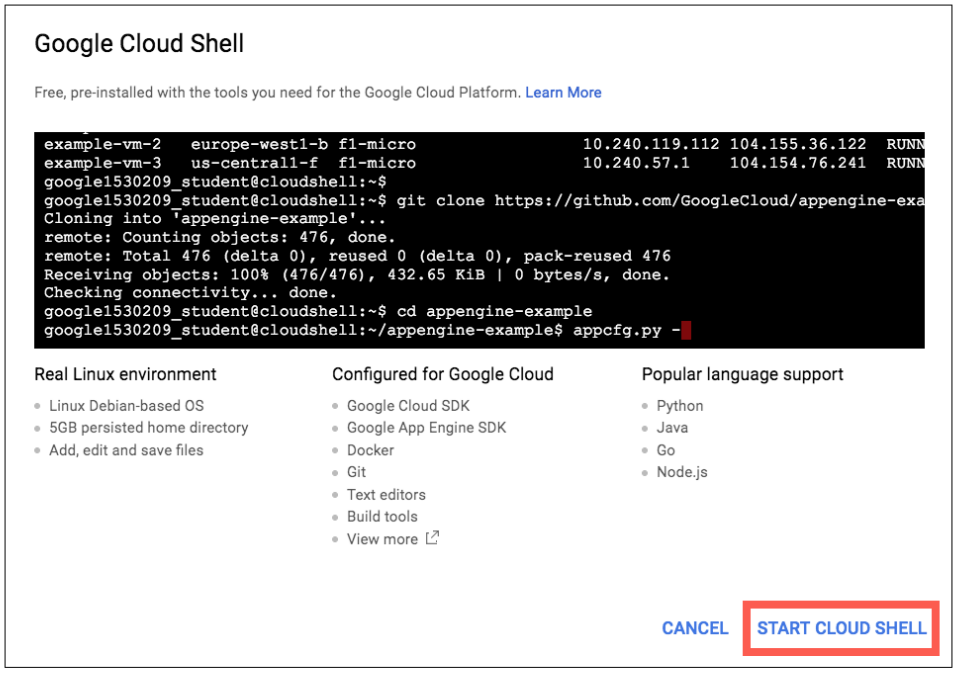
Activate Google Cloud Shell

Google 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. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



2. In the dialog box that opens, click **START CLOUD SHELL**:



Google Cloud Shell

Free, pre-installed with the tools you need for the Google Cloud Platform. Learn More

```
example-vm-2   europe-west1-b f1-micro                 10.240.119.112 104.155.36.122  RUNN
example-vm-3   us-central1-f  f1-micro                 10.240.57.1    104.154.76.241  RUNN
google1530209_student@cloudshell:~$
google1530209_student@cloudshell:~$ git clone https://github.com/GoogleCloud/appengine-exa
Cloning into 'appengine-example'...
remote: Counting objects: 476, done.
remote: Total 476 (delta 0), reused 0 (delta 0), pack-reused 476
Receiving objects: 100% (476/476), 432.65 KiB | 0 bytes/s, done.
Checking connectivity... done.
google1530209_student@cloudshell:~$ cd appengine-example
google1530209_student@cloudshell:~/appengine-example$ appcfg.py -
```

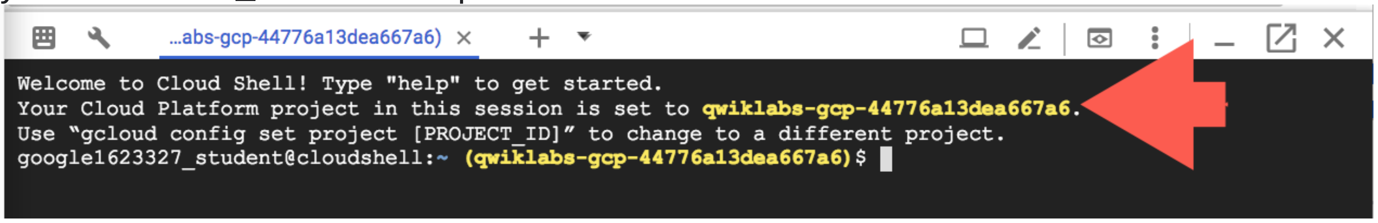
Real Linux environment
- Linux Debian-based OS
- 5GB persisted home directory
- Add, edit and save files

Configured for Google Cloud
- Google Cloud SDK
- Google App Engine SDK
- Docker
- Git
- Text editors
- Build tools
- View more

Popular language support
- Python
- Java
- Go
- Node.js

You can click "START CLOUD SHELL" immediately when the dialog box opens.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6)$
```

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

You can list the active account name with this command:

```
gcloud auth list
```

Output:

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

Example output:

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

Output:

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

Example output:

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

Full documentation of **gcloud** is available on Google Cloud gcloud Overview.

Set up your environment

In this section, you configure the infrastructure and identities required to complete the lab.

First, make sure the following APIs are enabled in the console:

1. Kubernetes Engine (GKE)
2. Cloud Build
3. Cloud Source Repositories
4. Cloud Container Registry
5. Cloud Pub/Sub.

On the **Navigation menu** (≡), click **APIs & services**.
Scroll down and confirm that your APIs are enabled.

If an API is missing, click **ENABLE APIS AND SERVICES** at the top, search for the API by name, and enable it for your project.

Create a GKE cluster

First, you'll create a GKE cluster to deploy Spinnaker and the sample app. You run all the terminal commands in this lab from Cloud Shell.

1. Set the default compute zone where your cluster instances will run:

```
gcloud config set compute/zone us-central1-f
```

3. Create a GKE cluster using the Spinnaker tutorial sample application:

```
gcloud container clusters create spinnaker-tutorial \
    --machine-type=n1-standard-2
```

This deployment will take between **five and ten minutes** to complete. You may see warnings including one about default scopes that you can safely ignore as they should have no impact on this lab. You should wait for the deployment to complete before proceeding.
When completed you'll see a report detailing the name, location, version, ip-address, machine-type, node version, number of nodes and status of the cluster indicating that the cluster is running.

Configure identity and access management

You create a Cloud Identity and Access Management (Cloud IAM) service account to delegate permissions to Spinnaker, allowing it to store data in Cloud Storage. Spinnaker stores its pipeline data in Cloud Storage to ensure reliability and resiliency. If your Spinnaker deployment unexpectedly fails, you can create an identical deployment in minutes with access to the same pipeline data as the original.

1. Create the service account:

```
gcloud iam service-accounts create spinnaker-account \
    --display-name spinnaker-account
```

4. Store the service account email address and your current project ID in environment variables for use in later commands:

```
export SA_EMAIL=$(gcloud iam service-accounts list \
    --filter="displayName:spinnaker-account" \
    --format='value(email)')
export PROJECT=$(gcloud info --format='value(config.project)')
```

9. Bind the `storage.admin` role to your service account:

```
gcloud projects add-iam-policy-binding $PROJECT \
    --role roles/storage.admin \
    --member serviceAccount:$SA_EMAIL
```

13. Download the service account key. You need this key later when you install Spinnaker and upload the key to GKE.

```
gcloud iam service-accounts keys create spinnaker-sa.json \
    --iam-account $SA_EMAIL
```

Set up Cloud Pub/Sub to trigger Spinnaker pipelines

1. Create the Cloud Pub/Sub topic for notifications from Container Registry. This command may fail with the error "Resource already exists in the project", which means that the topic has already been created for you.

```
gcloud pubsub topics create projects/$PROJECT/topics/gcr
```

3. Create a subscription that Spinnaker can read from to receive notifications of images being pushed.

```
gcloud pubsub subscriptions create gcr-triggers \
    --topic projects/${PROJECT}/topics/gcr
```

6. Give Spinnaker's service account permissions to read from the `gcr-triggers` subscription.

```
export SA_EMAIL=$(gcloud iam service-accounts list \
    --filter="displayName:spinnaker-account" \
    --format='value(email)')
gcloud beta pubsub subscriptions add-iam-policy-binding gcr-triggers \
    --role roles/pubsub.subscriber --member serviceAccount:$SA_EMAIL
```

Click *Check my progress* to verify the objective.

Set up the environment

Check my progress

Deploying Spinnaker using Helm

In this section, you use Helm to deploy Spinnaker from the Charts repository. Helm is a package manager you can use to configure and deploy Kubernetes apps.

Install Helm

1. Download and install the `helm` binary:

```
wget  https://storage.googleapis.com/kubernetes-helm/helm-v2.10.0-linux-amd64.tar.gz
```

3. Unzip the file to your local system:

```
tar zxfv helm-v2.10.0-linux-amd64.tar.gz
cp linux-amd64/helm .
```

6. Grant Tiller, the server side of Helm, the cluster-admin role in your cluster:

```
kubectl create clusterrolebinding user-admin-binding \
    --clusterrole=cluster-admin --user=$(gcloud config get-value account)
kubectl create serviceaccount tiller \
    --namespace kube-system
kubectl create clusterrolebinding tiller-admin-binding \
    --clusterrole=cluster-admin --serviceaccount=kube-system:tiller
```

13. Grant Spinnaker the `cluster-admin` role so it can deploy resources across all namespaces:

```
kubectl create clusterrolebinding --clusterrole=cluster-admin \
    --serviceaccount=default:default spinnaker-admin
```

16. Initialize Helm to install Tiller in your cluster:

```
./helm init --service-account=tiller
./helm update
```

19. Ensure that Helm is properly installed by running the following command. If Helm is correctly installed, `v2.10.0` appears for both client and server.

```
./helm version
```

Output (do not copy)

```
Client: &version.Version{SemVer:"v2.10.0", GitCommit:"9ad53aac42165a5fadc6c87be0dea6b115f93090", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.10.0", GitCommit:"9ad53aac42165a5fadc6c87be0dea6b115f93090", GitTreeState:"clean"}
```

Configure Spinnaker

1. Create a bucket for Spinnaker to store its pipeline configuration:

```
export PROJECT=$(gcloud info \
    --format='value(config.project)')
export BUCKET=$PROJECT-spinnaker-config
gsutil mb -c regional -l us-central1 gs://$BUCKET
```

6. Run the following command to create a `spinnaker-config.yaml` file, which describes how Helm should install Spinnaker:

```
export SA_JSON=$(cat spinnaker-sa.json)
export PROJECT=$(gcloud info --format='value(config.project)')
export BUCKET=$PROJECT-spinnaker-config
cat > spinnaker-config.yaml <<EOF
gcs:
  enabled: true
  bucket: $BUCKET
  project: $PROJECT
  jsonKey: '$SA_JSON'

dockerRegistries:
- name: gcr
  address: https://gcr.io
  username: _json_key
  password: '$SA_JSON'
  email: 1234@5678.com

# Disable minio as the default storage backend
minio:
  enabled: false

# Configure Spinnaker to enable GCP services
halyard:
  spinnakerVersion: 1.10.2
  image:
    tag: 1.12.0
  additionalScripts:
    create: true
    data:
      enable_gcs_artifacts.sh: |-
        \$HAL_COMMAND config artifact gcs account add gcs-$PROJECT --json-path /opt/gcs/key.json
        \$HAL_COMMAND config artifact gcs enable
      enable_pubsub_triggers.sh: |-
        \$HAL_COMMAND config pubsub google enable
        \$HAL_COMMAND config pubsub google subscription add gcr-triggers \
          --subscription-name gcr-triggers \
          --json-path /opt/gcs/key.json \
          --project $PROJECT \
          --message-format GCR
EOF
```

Deploy the Spinnaker chart

1. Use the Helm command-line interface to deploy the chart with your configuration set. This command typically takes **five to ten minutes** to complete.

```
./helm install -n cd stable/spinnaker -f spinnaker-config.yaml \
    --timeout 600 --version 1.1.6 --wait
```

Click *Check my progress* to verify the objective.

Deploying Spinnaker using Helm

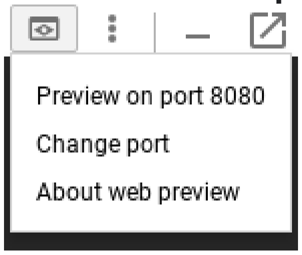
Check my progress

2. After the command completes, run the following command to set up port forwarding to the Spinnaker UI from Cloud Shell:

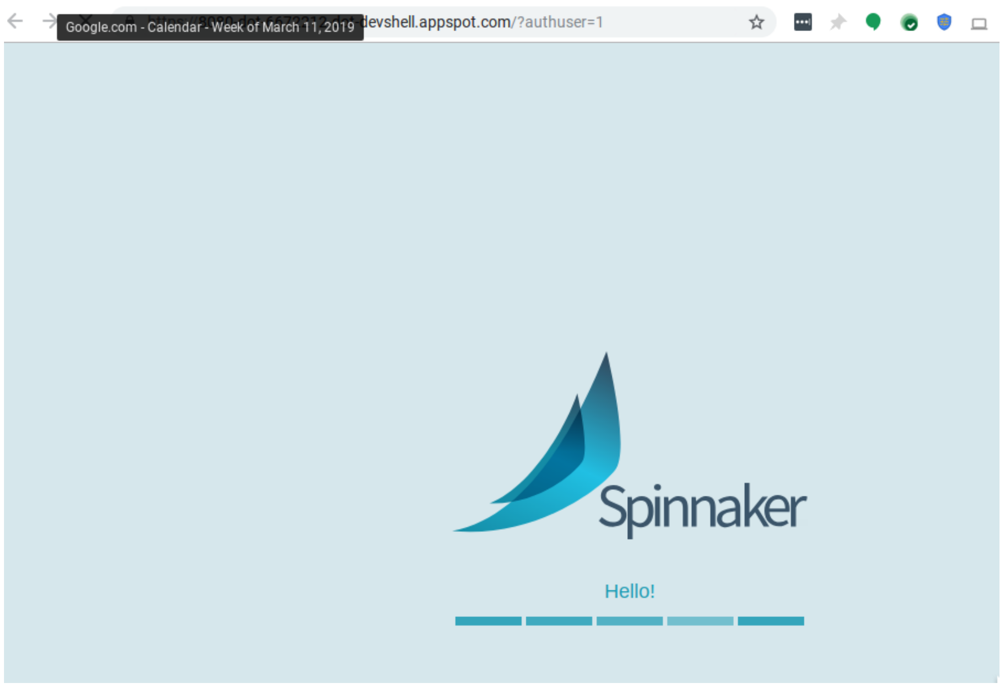
```
export DECK_POD=$(kubectl get pods --namespace default -l "cluster=spin-deck" \
    -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward --namespace default $DECK_POD 8080:9000 >> /dev/null &
```

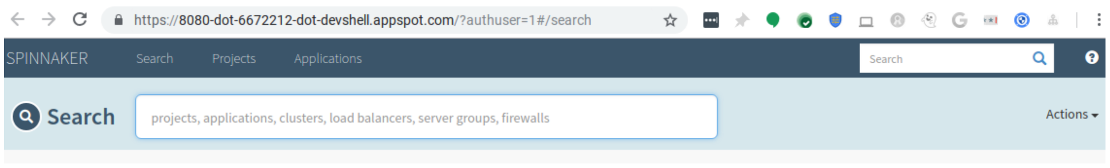
Note: This command can take several minutes to complete. Be sure to wait until you see that it has succeeded before proceeding.

6. To open the Spinnaker user interface, click **Web Preview** in Cloud Shell and click **Preview on port 8080**.



7. You should see the welcome screen, followed by the Spinnaker UI:


Google.com - Calendar - Week of March 11, 2019
devshell.appspot.com/?authuser=1
Spinnaker
Hello!


https://8080-dot-6672212-dot-devshell.appspot.com/?authuser=1#/search
SPINNAKER
Search
Projects
Applications
Search
Search
projects, applications, clusters, load balancers, server groups, firewalls
Actions

Building the Docker image

In this section, you configure Cloud Build to detect changes to your app source code, build a Docker image, and then push it to Container Registry.

Create your source code repository

1. In Cloud Shell, download the sample source code:

```
2. wget https://gke-spinnaker.storage.googleapis.com/sample-app-v2.tgz
```

3. Unpack the source code:

```
4. tar xzfv sample-app-v2.tgz
```

5. Change directories to the source code:

```
6. cd sample-app
```

7. Set the username and email address for your Git commits in this repository. You can replace `[USERNAME]` with a username you create.

```
8. git config --global user.email "$(gcloud config get-value core/account)"
9. git config --global user.name "[USERNAME]"
```

10. Make the initial commit to your source code repository:

```
11.  git init
12.  git add .
13.  git commit -m "Initial commit"
```

14. Create a repository to host your code:

```
15.  gcloud source repos create sample-app
16.  git config credential.helper gcloud.sh
```

17. Add your newly created repository as remote:

```
18.  export PROJECT=$(gcloud info --format='value(config.project)')
19.  git                  remote                   add                  origin
   https://source.developers.google.com/p/$PROJECT/r/sample-app
```

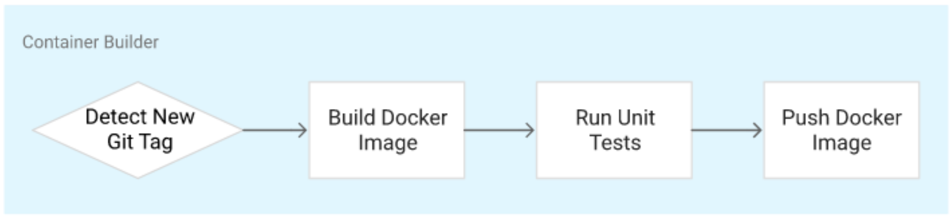
20. Push your code to the new repository's master branch:

```
21.  git push origin master
```

22. Check that you can see your source code in the console by clicking **Navigation menu**, then in the **Tools** section, click **Source Repositories > sample-app**.

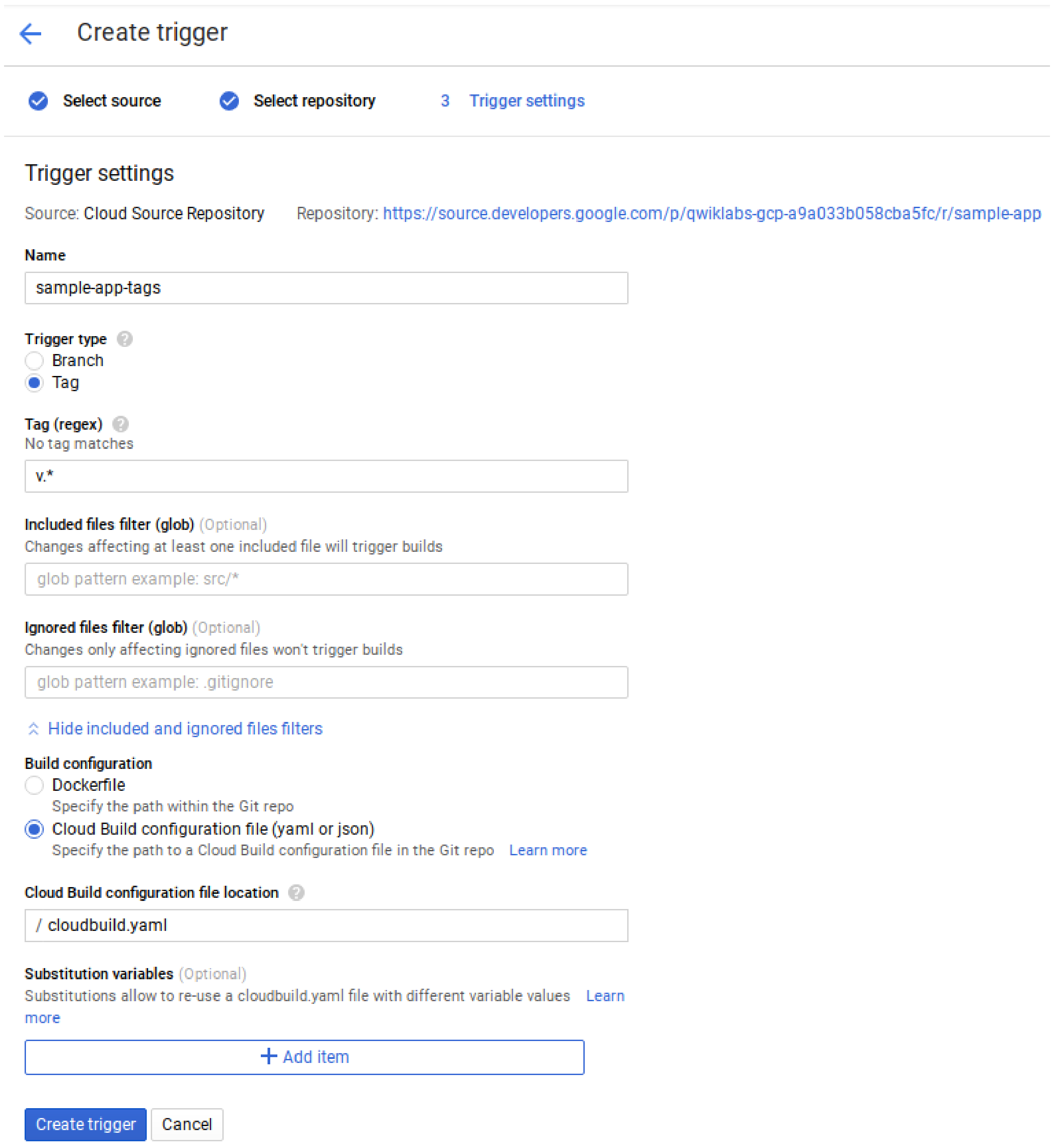
Configure your build triggers

In this section, you configure Cloud Build to build and push your Docker images every time you push Git tags to your source repository. Cloud Build automatically checks out your source code, builds the Docker image from the Dockerfile in your repository, and pushes that image to Container Registry.



1. In the GCP Console, go again to the **Tools** section of the **Navigation menu**, then click **Cloud Build > Triggers**.
2. Click **Create trigger**.
3. Select **Cloud Source Repository** and click **Continue**.
4. Select your newly created `sample-app` repository from the list, and click **Continue**.
5. Set the following trigger settings:

   - **Name**: `sample-app-tags`
   - **Trigger type**: Tag
   - **Tag (regex)**: `v.*`
   - **Build configuration**: `Cloud Build configuration file (yaml or json)`
   - **Cloud Build configuration file location**: `cloudbuild.yaml`
6. Click **Create trigger**.


Create trigger

Select source   Select repository   3 Trigger settings

Trigger settings

Source: Cloud Source Repository   Repository: https://source.developers.google.com/p/qwiklabs-gcp-a9a033b058cba5fc/r/sample-app

Name

sample-app-tags

Trigger type

Branch

[✓ Tag]

Tag (regex)

No tag matches

v.*

Included files filter (glob) (Optional)

Changes affecting at least one included file will trigger builds

glob pattern example: src/*

Ignored files filter (glob) (Optional)

Changes only affecting ignored files won't trigger builds

glob pattern example: .gitignore

Hide included and ignored files filters

Build configuration

Dockerfile

Specify the path within the Git repo

[✓ Cloud Build configuration file (yaml or json)]

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

Cloud Build configuration file location

/ cloudbuild.yaml

Substitution variables (Optional)

Substitutions allow to re-use a cloudbuild.yaml file with different variable values   Learn more

+ Add item

Create trigger   Cancel

From now on, whenever you push a Git tag prefixed with the letter "v" to your source code repository, Cloud Build automatically builds and pushes your app as a Docker image to Container Registry.

Prepare your Kubernetes Manifests for use in Spinnaker

Spinnaker needs access to your Kubernetes manifests in order to deploy them to your clusters. This section creates a Cloud Storage bucket that will be populated with your manifests during the CI process in Cloud Build. After your manifests are in Cloud Storage, Spinnaker can download and apply them during your pipeline's execution.

1. Create the bucket.

```
2. export PROJECT=$(gcloud info --format='value(config.project)')
3. gsutil mb -l us-central1 gs://$PROJECT-kubernetes-manifests
```

4. Enable versioning on the bucket so that you have a history of your manifests.

```
5. gsutil versioning set on gs://$PROJECT-kubernetes-manifests
```

6. Set the correct project ID in your kubernetes deployment manifests:

```
7. sed -i s/PROJECT/$PROJECT/g k8s/deployments/*
```

8. Commit the changes to the repository:

```
9. git commit -a -m "Set project ID"
```

Build your image

Push your first image using the following steps:

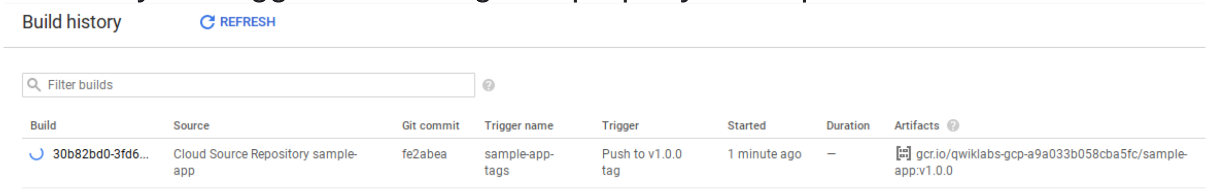
1. From your `sample-app` source code directory in Cloud Shell.
2. Create a Git tag:

```
3. git tag v1.0.0
```

4. Push the tag:

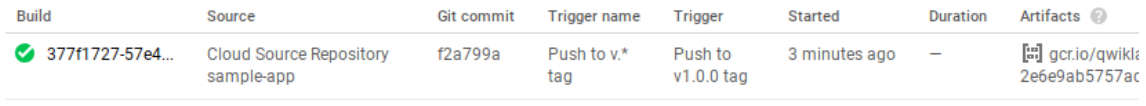
```
5. git push --tags
```

6. In **Cloud Build**, click **History** to check that the build has been triggered. If not, verify the trigger was configured properly in the previous section.



Build	Source	Git commit	Trigger name	Trigger	Started	Duration	Artifacts
30b82bd0-3fd6...	Cloud Source Repository sample-app	fe2abea	sample-app-tags	Push to v1.0.0 tag	1 minute ago	–	gcr.io/qwiklabs-gcp-a9a033b058cba5fc/sample-app:v1.0.0

7. **Stay on this page and wait** for the build to complete before going on to the next step.



Build	Source	Git commit	Trigger name	Trigger	Started	Duration	Artifacts
✓ 377f1727-57e4...	Cloud Source Repository sample-app	f2a799a	Push to v.* tag	Push to v1.0.0 tag	3 minutes ago	–	gcr.io/qwikla 2e6e9ab5757ac

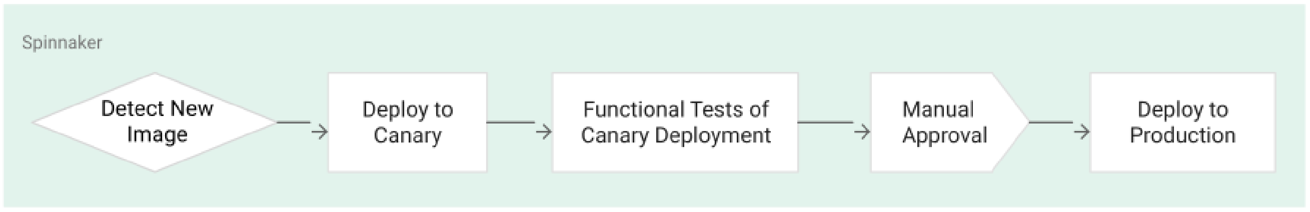
Click *Check my progress* to verify the objective.
Building the Docker image

Check my progress

Configuring your deployment pipelines

Now that your images are building automatically, you need to deploy them to the Kubernetes cluster.

You deploy to a scaled-down environment for integration testing. After the integration tests pass, you must manually approve the changes to deploy the code to production services.



Install the spin CLI for managing Spinnaker

[spin](#) is a command-line utility for managing Spinnaker's applications and pipelines.

1. Download the 1.6.0 version of `spin`.

```
curl -LO https://storage.googleapis.com/spinnaker-artifacts/spin/1.6.0/linux/amd64/spin
```

3. Make `spin` executable.

```
chmod +x spin
```

Create the deployment pipeline

1. Use `spin` to create an app called **sample** in Spinnaker. Set the owner email address for the app in Spinnaker.

```
./spin application save --application-name sample \
                        --owner-email "$(gcloud config get-value core/account)" \
                        --cloud-providers kubernetes \
                        --gate-endpoint http://localhost:8080/gate
```

Next, you create the continuous delivery pipeline. In this tutorial, the pipeline is configured to detect when a Docker image with a tag prefixed with "v" has arrived in your Container Registry.

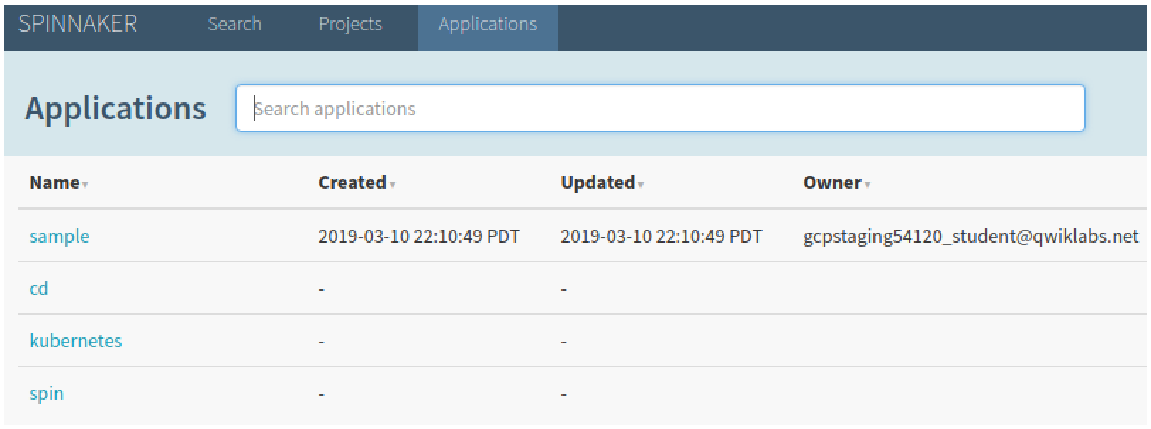
2. Open a **new tab** in Cloud Shell using **Add Cloud Shell session**.
3. Return to your `sample-app` source code directory and run the following command to upload an example pipeline to your Spinnaker instance:

```
export PROJECT=$(gcloud info --format='value(config.project)')
sed s/PROJECT/$PROJECT/g spinnaker/pipeline-deploy.json > pipeline.json
./spin pipeline save --gate-endpoint http://localhost:8080/gate -f pipeline.json
```

Manually Trigger and View your pipeline execution

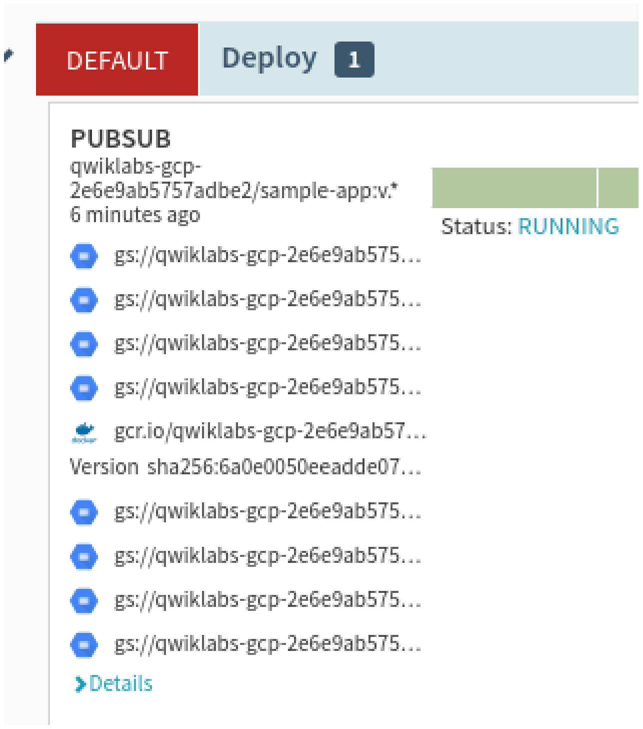
The configuration you just created uses notifications of newly tagged images being pushed to trigger a Spinnaker pipeline. In a previous step, you pushed a tag to the Cloud Source Repositories which triggered Cloud Build to build and push your image to Container Registry. You can now **manually trigger** that pipeline to verify it.

1. Return to the Spinnaker UI and click **Applications** at the top of the screen to see your list of managed applications. **sample** is your application. If you don't see `sample`, try refreshing the Spinnaker Applications tab.

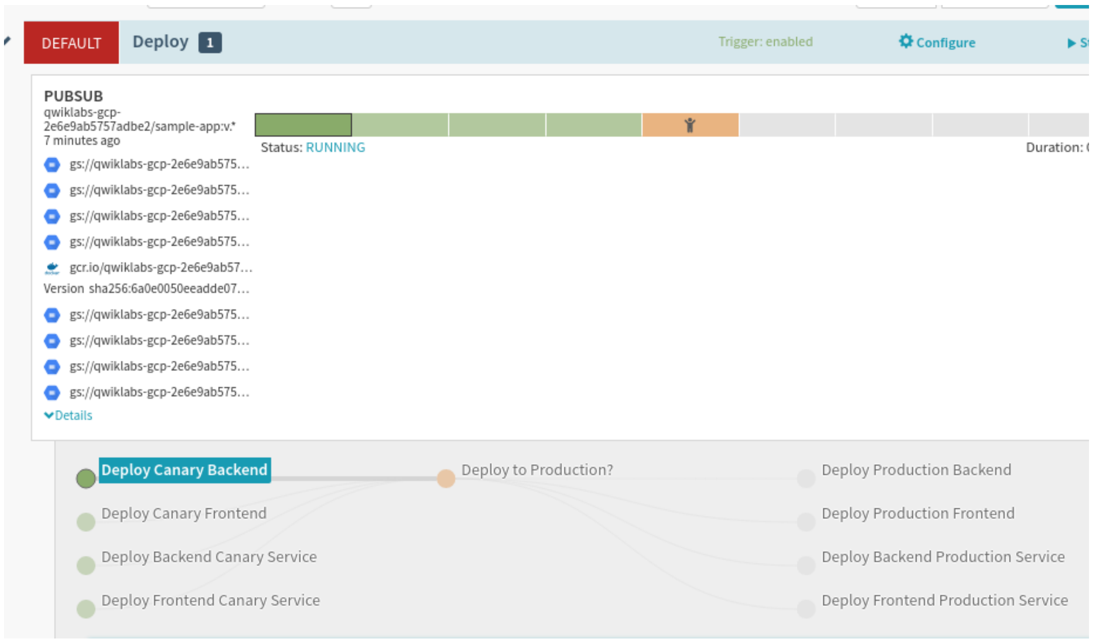


Name	Created	Updated	Owner
sample	2019-03-10 22:10:49 PDT	2019-03-10 22:10:49 PDT	gcpstaging54120_student@qwiklabs.net
cd	-	-	
kubernetes	-	-	
spin	-	-	

2. Click **sample** to view your application deployment.
3. Click **Pipelines** at the top to view your applications pipeline status.
4. Click **Start Manual Execution** to trigger the pipeline this first time. Click **Run**.
5. Click **Details** to see more information about the pipeline's progress. This section shows the status of the deployment pipeline and its steps. Steps in blue are currently running, green ones have completed successfully, and red ones have failed.

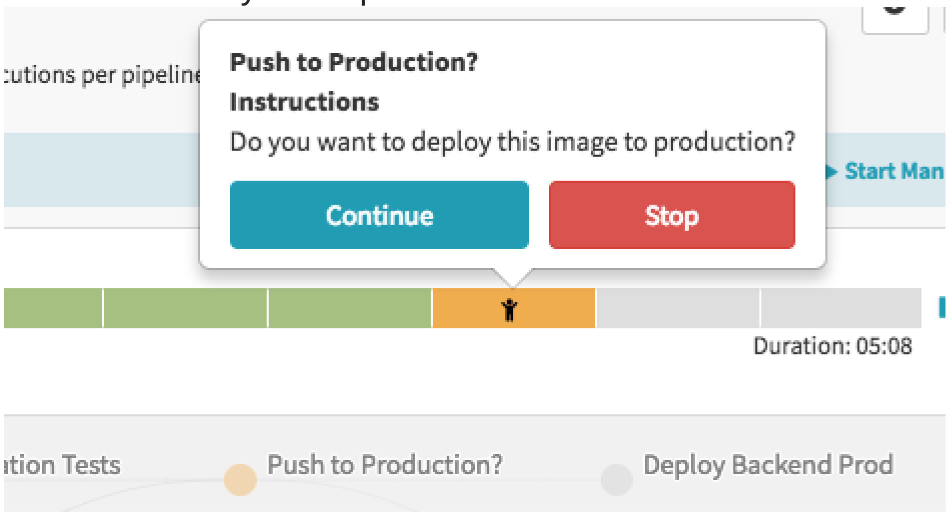


PUBSUB
qwiklabs-gcp-2e6e9ab5757adbe2/sample-app:v.*
6 minutes ago
Status: RUNNING
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
gcr.io/qwiklabs-gcp-2e6e9ab57…
Version sha256:6a0e0050eeadde07…
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
gs://qwiklabs-gcp-2e6e9ab575…
Details


DEFAULT
Deploy 1
Trigger: enabled
Configure
PUBSUB
qwiklabs-gcp-2e6e9ab5757adbe2/sample-app:v.*
7 minutes ago
Status: RUNNING
gs://qwiklabs-gcp-2e6e9ab575...
gcr.io/qwiklabs-gcp-2e6e9ab57...
Version sha256:6a0e0050eeadde07...
Details
Deploy Canary Backend
Deploy to Production?
Deploy Production Backend
Deploy Canary Frontend
Deploy Production Frontend
Deploy Backend Canary Service
Deploy Backend Production Service
Deploy Frontend Canary Service
Deploy Frontend Production Service

6. Click a stage to see details about it.

   After **3 to 5 minutes** the integration test phase completes and the pipeline requires manual approval to continue the deployment.
7. Hover over the yellow "person" icon and click **Continue**.



   Your rollout continues to the production frontend and backend deployments. It completes after a few minutes.

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

Create service load balancers

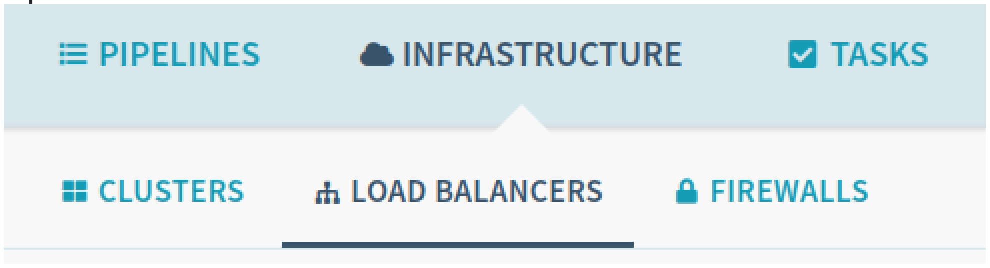
Check my progress

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

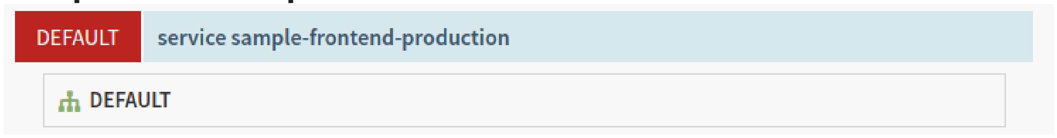
Deploy image to production

Check my progress

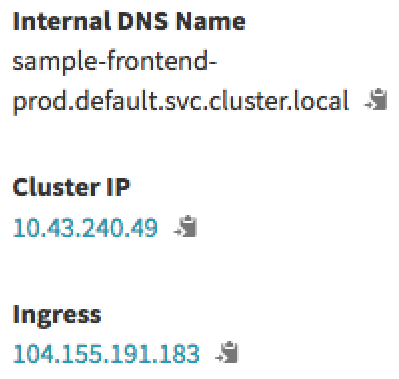
8. To view the app, select **Infrastructure** > **Load Balancers** in the top of the Spinnaker UI.



9. Scroll down the list of load balancers and click **Default**, under **service sample-frontend-production**.



10. Scroll down the details pane on the right and copy your app's IP address by clicking the clipboard button on the **Ingress** IP. The ingress IP link from the Spinnaker UI may use HTTPS by default, while the application is configured to use HTTP.

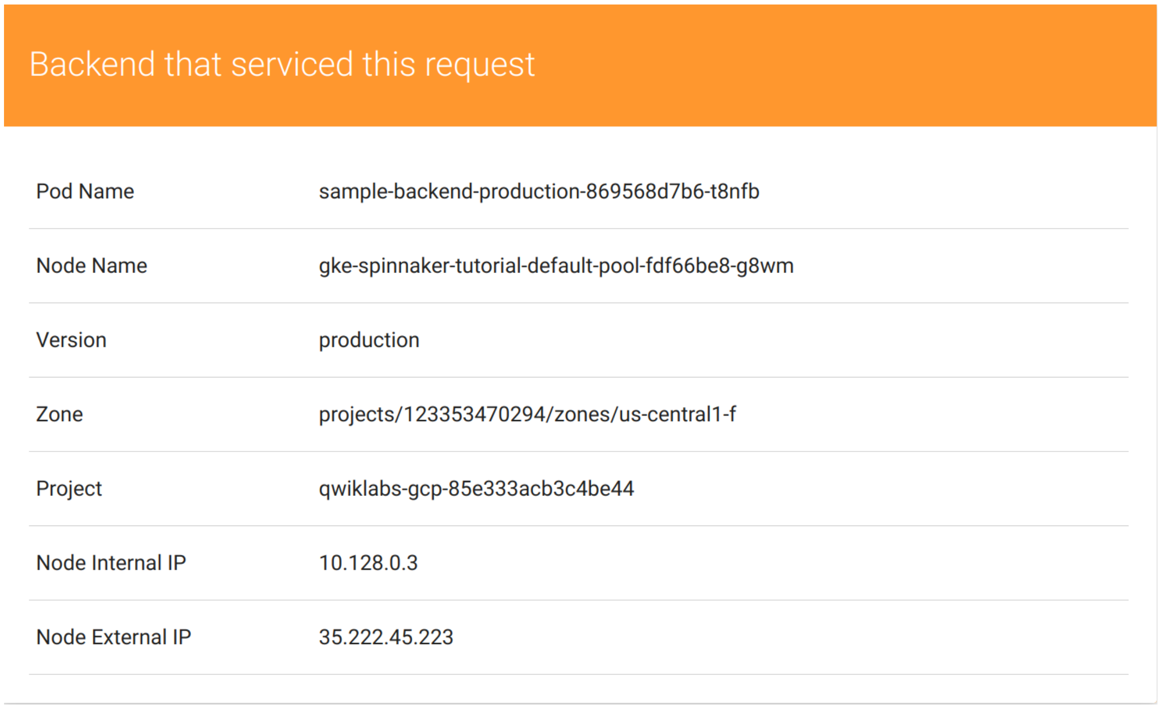


Internal DNS Name
sample-frontend-prod.default.svc.cluster.local

Cluster IP
10.43.240.49

Ingress
104.155.191.183

11. Paste the address into your browser to view the production version of the app. You may need to make sure it is an HTTP address.



Backend that serviced this request

Pod Name	sample-backend-production-869568d7b6-t8nfb
Node Name	gke-spinnaker-tutorial-default-pool-fdf66be8-g8wm
Version	production
Zone	projects/123353470294/zones/us-central1-f
Project	qwiklabs-gcp-85e333acb3c4be44
Node Internal IP	10.128.0.3
Node External IP	35.222.45.223

You have now manually triggered the pipeline to build, test, and deploy your app.

Triggering your pipeline from code changes

In this section, you test the pipeline end to end by making a code change, pushing a Git tag, and watching the pipeline run in response. By pushing a Git tag that starts with "v", you trigger Cloud Build to build a new Docker image and push it to Container Registry. Spinnaker detects that the new image tag begins with "v" and triggers a pipeline to deploy the image to canaries, run tests, and roll out the same image to all pods in the deployment.

1. From your `sample-app` directory, change the color of the app from orange to blue:

```
2. sed -i 's/orange/blue/g' cmd/gke-info/common-service.go
```

3. Tag your change and push it to the source code repository:

```
4. git commit -a -m "Change color to blue"
5. git tag v1.0.1
6. git push --tags
```
7. In the console, open **Cloud Build > History** and wait a couple minutes for the new build to appear. You may need to refresh the page.
8. Return to the Spinnaker UI and click **Pipelines** to watch the pipeline start to deploy the image.

Click *Check my progress* to verify the objective.

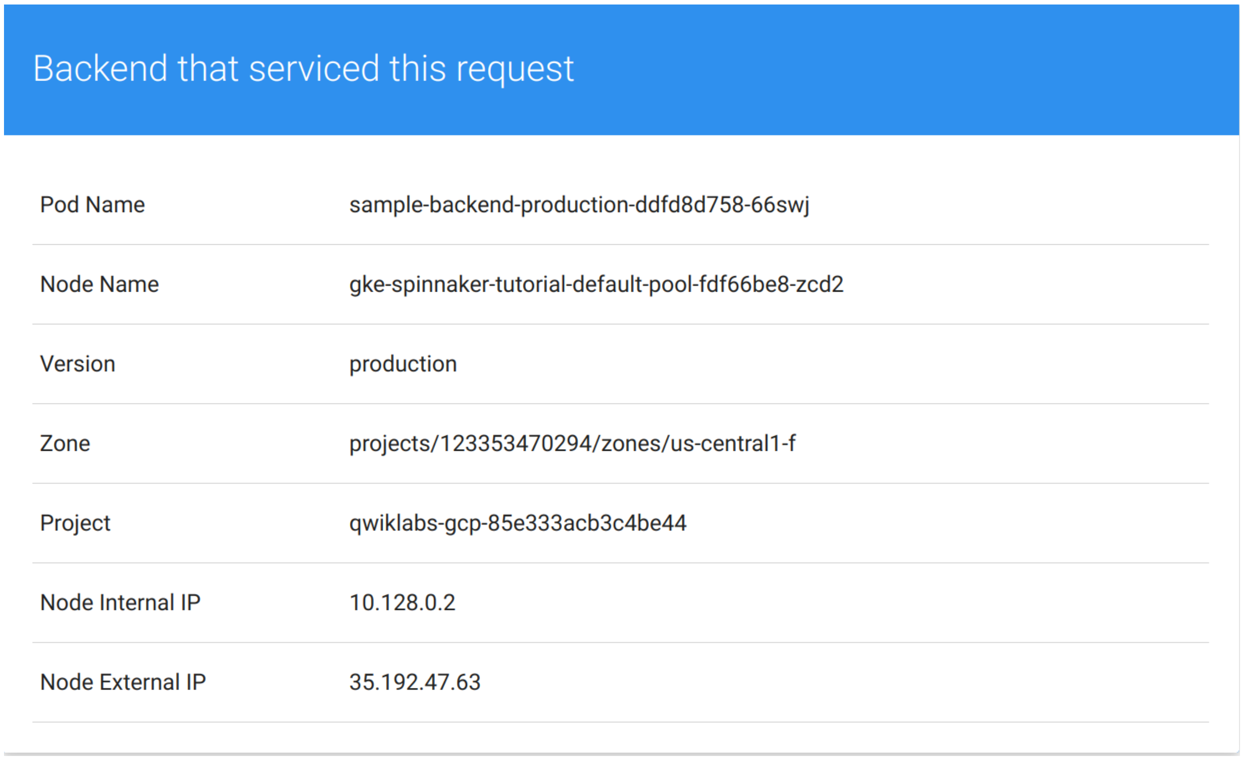
Triggering pipeline from code changes

Check my progress

Observe the canary deployments

1. When the deployment is paused, waiting to roll out to production, return to the web page displaying your running application and start refreshing the tab that contains your app. Four of your backends are running the previous version of your app, while only one backend is running the canary. You should see the new, blue version of your app appear about every fifth time you refresh.

2. After testing completes, return to the **Spinnaker** tab and approve the deployment by clicking **Continue**.
3. When the pipeline completes, your app looks like the following screenshot. Note that the color has changed to blue because of your code change, and that the **Version** field now reads `production`.



Backend that serviced this request

Pod Name	sample-backend-production-ddfd8d758-66swj
Node Name	gke-spinnaker-tutorial-default-pool-fdf66be8-zcd2
Version	production
Zone	projects/123353470294/zones/us-central1-f
Project	qwiklabs-gcp-85e333acb3c4be44
Node Internal IP	10.128.0.2
Node External IP	35.192.47.63

You have now successfully rolled out your app to your entire production environment!

4. Optionally, you can roll back this change by reverting your previous commit. Rolling back adds a new tag `(v1.0.2)`, and pushes the tag back through the same pipeline you used to deploy `v1.0.1`:

```
5. git revert v1.0.1
6. git tag v1.0.2
7. git push --tags
```

When the pipeline is complete, to verify the roll back, click on **Infrastructure > Load Balancers**, then **service sample-frontend-canary**, click **Default** and copy the Ingress IP address into a new tab.
Now your app is back to orange and you can see the `canary` version number.