Continuous Deployment with Spinnaker v1.6

3 hours9 Credits

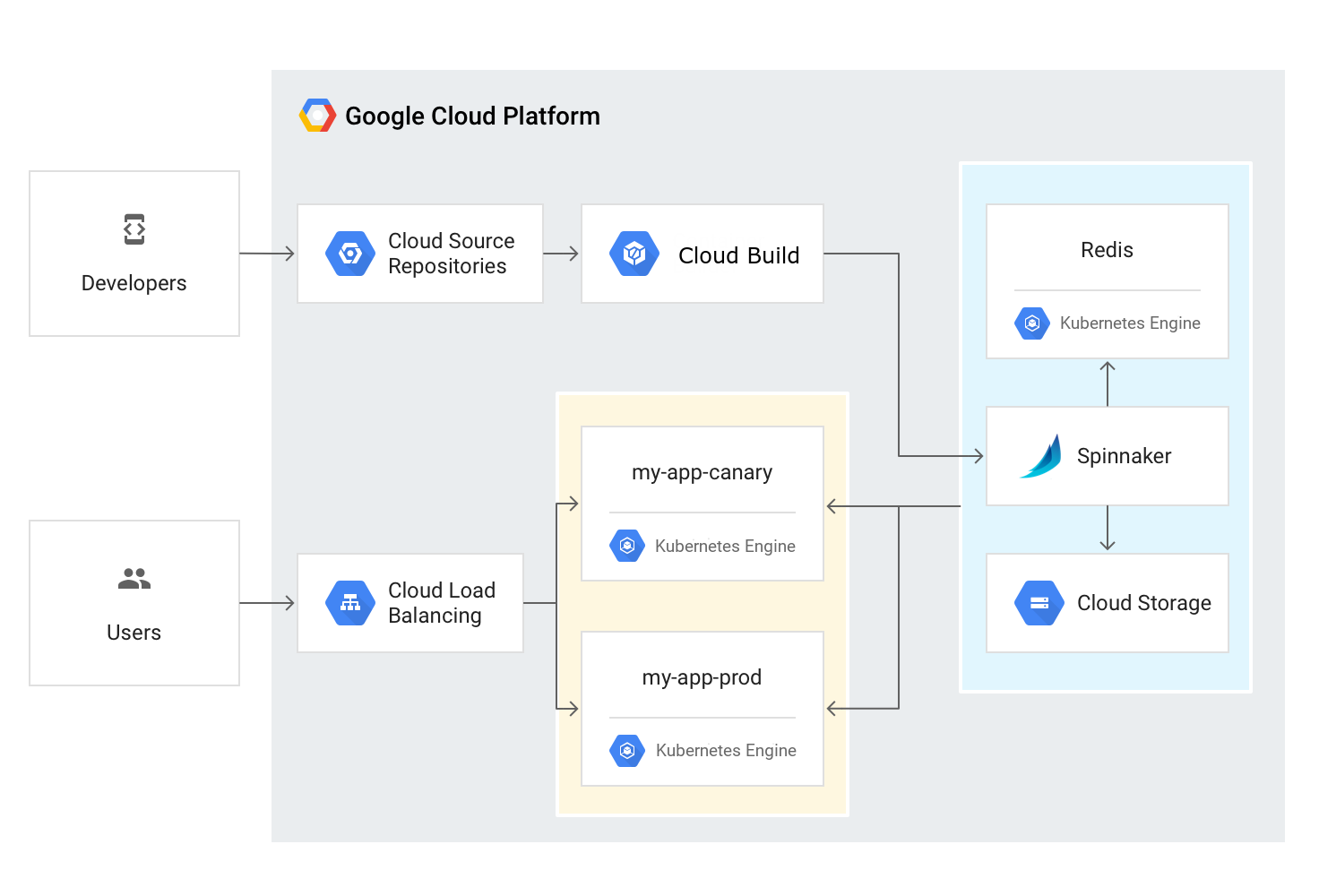
Rate Lab

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](https://cloud.google.com/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](https://github.com/kubernetes/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**



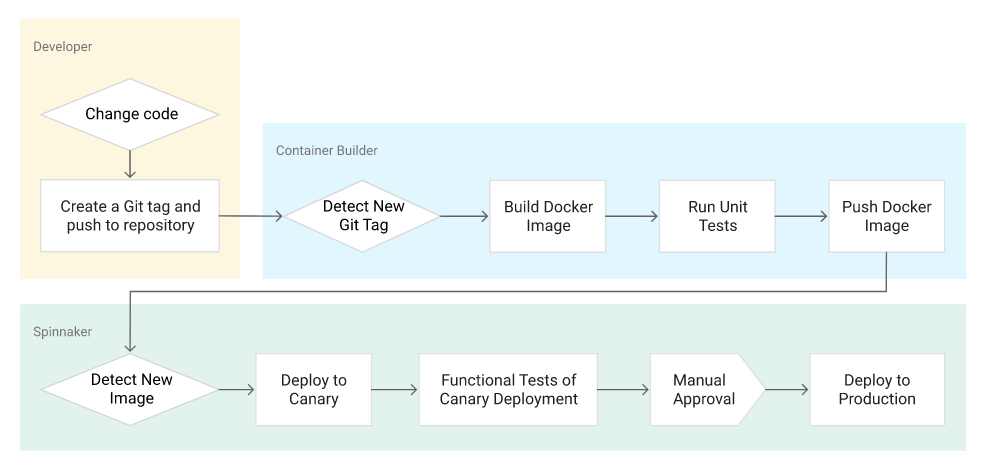
To continuously deliver app updates to your users, you need an automated process that reliably builds, tests, and updates your software. **Code changes should automatically flow through a pipeline that includes artifact creation, unit testing, functional testing, and production rollout**. In some cases, 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](https://martinfowler.com/bliki/CanaryRelease.html)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

**What you'll need**

To complete this lab, you’ll need:

* Access to a standard internet browser (Chrome browser recommended).
* Time. Note the lab’s **Completion** time in Qwiklabs. This is an estimate of the time it should take to complete all steps. Plan your schedule so you have time to complete the lab. Once you start the lab, you will not be able to pause and return later (you begin at step 1 every time you start a lab).
* The lab's **Access** time is how long your lab resources will be available. If you finish your lab with access time still available, you will be able to explore the Google Cloud Platform or work on any section of the lab that was marked "if you have time". Once the Access time runs out, your lab will end and all resources will terminate.
* You **DO NOT** need a Google Cloud Platform account or project. An account, project and associated resources are provided to you as part of this lab.
* If you already have your own GCP account, make sure you do not use it for this lab.
* If your lab prompts you to log into the console, **use only the student account provided to you by the lab**. This prevents you from incurring charges for lab activities in your personal GCP account.

**Start your lab**

When you are ready, click **Start Lab**. You can track your lab’s progress with the status bar at the top of your screen.

**Important** What is happening during this time? Your lab is spinning up GCP resources for you behind the scenes, including an account, a project, resources within the project, and permission for you to control the resources needed to run the lab. This means that instead of spending time manually setting up a project and building resources from scratch as part of your lab, you can begin learning more quickly.

**Find Your Lab’s GCP Username and Password**

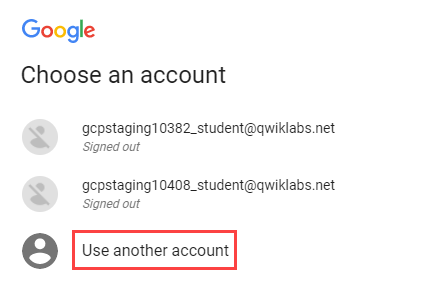
To access the resources and console for this lab, locate the Connection Details panel in Qwiklabs. Here you will find the account ID and password for the account you will use to log in to the Google Cloud Platform:



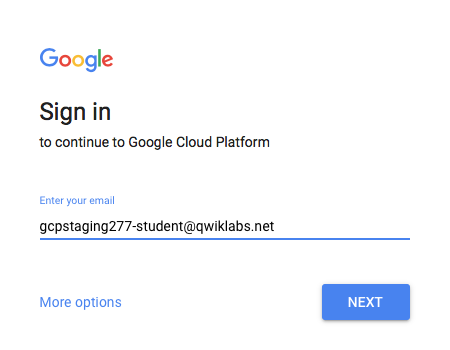
If your lab provides other resource identifiers or connection-related information, it will appear on this panel as well.

**Log in to Google Cloud Console**

Using the Qwiklabs browser tab/window or the separate browser you are using for the Qwiklabs session, copy the Username from the Connection Details panel and click the **Open Google Console** button.

You'll be asked to Choose an account. Click **Use another account**. 

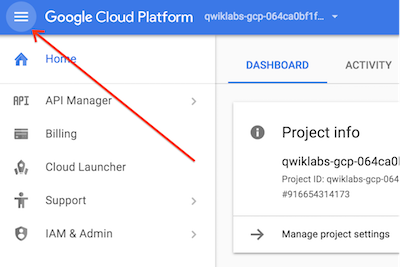
Paste in the Username, and then the Password as prompted:



Accept the terms and conditions.

Since this is a temporary account, which you will only have to access for this one lab:

* Do not add recovery options
* Do not sign up for free trials

**Note:** You can view the list of services by clicking the GCP Navigation menu button at the top-left next to “Google Cloud Platform”.

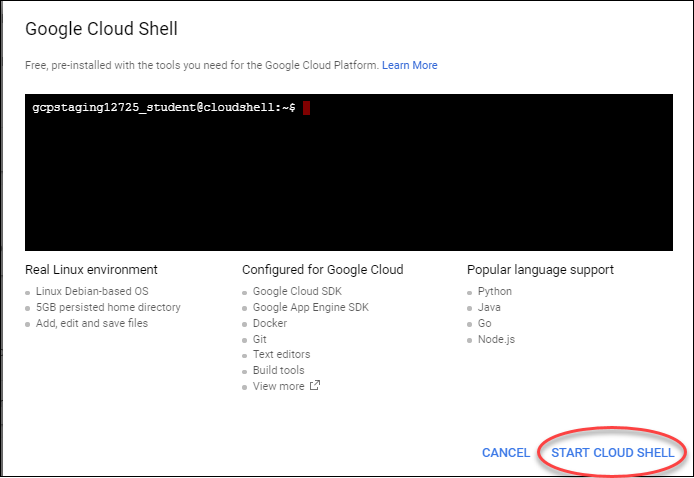
Activate Google Cloud Shell

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

From the GCP Console click the **Cloud Shell** icon on the top right toolbar:

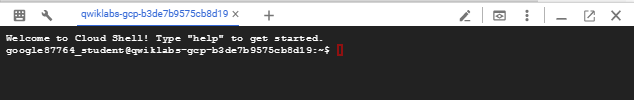


Then click **START CLOUD SHELL**:



You can click **START CLOUD SHELL** immediately when the dialog comes up instead of waiting in the dialog until the Cloud Shell provisions.

It takes a few moments to provision and connects to the environment:



The Cloud Shell is a virtual machine loaded with all the development tools you’ll need. It offers a persistent 5GB home directory, and runs on the Google Cloud, greatly enhancing network performance and authentication.

Once connected to the cloud shell, you'll see that you are already authenticated and the project is set to your *PROJECT\_ID*:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

**Note:** gcloud is the powerful and unified command-line tool for Google Cloud Platform. Full documentation is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud). It comes pre-installed on Cloud Shell and supports tab-completion.

gcloud config list project

Output:

[core]

project = <PROJECT\_ID>

**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** (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:
2. gcloud config set compute/zone us-central1-f
3. Create a GKE cluster using the Spinnaker tutorial sample application:
4. gcloud container clusters create spinnaker-tutorial \
5. --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](https://cloud.google.com/iam/docs/service-accounts) 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:
2. gcloud iam service-accounts create spinnaker-account \
3. --display-name spinnaker-account
4. Store the service account email address and your current project ID in environment variables for use in later commands:
5. export SA\_EMAIL=$(gcloud iam service-accounts list \
6. --filter="displayName:spinnaker-account" \
7. --format='value(email)')
8. export PROJECT=$(gcloud info --format='value(config.project)')
9. Bind the storage.admin role to your service account:
10. gcloud projects add-iam-policy-binding $PROJECT \
11. --role roles/storage.admin \
12. --member serviceAccount:$SA\_EMAIL
13. Download the service account key. You need this key later when you install Spinnaker and upload the key to GKE.
14. gcloud iam service-accounts keys create spinnaker-sa.json \
15. --iam-account $SA\_EMAIL

created key [b8ae352ca332a6312e43df6bed8cac77e48b9f13] of type [json] as [spinnaker-sa.json] for [spinnaker-account@qwiklabs-gcp-8f28ef47776fc4d1.iam.gserviceaccount.com]

google3614754\_student@cloudshell:~ (qwiklabs-gcp-8f28ef47776fc4d1)$

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.
2. gcloud pubsub topics create projects/$PROJECT/topics/gcr
3. Create a subscription that Spinnaker can read from to receive notifications of images being pushed.
4. gcloud pubsub subscriptions create gcr-triggers \
5. --topic projects/${PROJECT}/topics/gcr
6. Give Spinnaker's service account permissions to read from the gcr-triggers subscription.
7. export SA\_EMAIL=$(gcloud iam service-accounts list \
8. --filter="displayName:spinnaker-account" \
9. --format='value(email)')
10. gcloud beta pubsub subscriptions add-iam-policy-binding gcr-triggers \
11. --role roles/pubsub.subscriber --member serviceAccount:$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](https://github.com/helm/helm) to deploy Spinnaker from the [Charts](https://github.com/helm/charts)repository. Helm is a package manager you can use to configure and deploy [Kubernetes apps](https://hub.kubeapps.com/).

Install Helm

1. Download and install the helm binary:
2. wget https://storage.googleapis.com/kubernetes-helm/helm-v2.10.0-linux-amd64.tar.gz
3. Unzip the file to your local system:
4. tar zxfv helm-v2.10.0-linux-amd64.tar.gz
5. cp linux-amd64/helm .
6. Grant Tiller, the server side of Helm, the cluster-admin role in your cluster:
7. kubectl create clusterrolebinding user-admin-binding \
8. --clusterrole=cluster-admin --user=$(gcloud config get-value account)
9. kubectl create serviceaccount tiller \
10. --namespace kube-system
11. kubectl create clusterrolebinding tiller-admin-binding \
12. --clusterrole=cluster-admin --serviceaccount=kube-system:tiller
13. Grant Spinnaker the cluster-admin role so it can deploy resources across all namespaces:
14. kubectl create clusterrolebinding --clusterrole=cluster-admin \
15. --serviceaccount=default:default spinnaker-admin
16. Initialize Helm to install Tiller in your cluster:
17. ./helm init --service-account=tiller
18. ./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.
20. ./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:
2. export PROJECT=$(gcloud info \
3. --format='value(config.project)')
4. export BUCKET=$PROJECT-spinnaker-config
5. gsutil mb -c regional -l us-central1 gs://$BUCKET
6. Run the following command to create a spinnaker-config.yaml file, which describes how Helm should install Spinnaker:
7. export SA\_JSON=$(cat spinnaker-sa.json)
8. export PROJECT=$(gcloud info --format='value(config.project)')
9. export BUCKET=$PROJECT-spinnaker-config
10. cat > spinnaker-config.yaml <<EOF
11. gcs:
12. enabled: true
13. bucket: $BUCKET
14. project: $PROJECT
15. jsonKey: '$SA\_JSON'
16. dockerRegistries:
17. - name: gcr
18. address: https://gcr.io
19. username: \_json\_key
20. password: '$SA\_JSON'
21. email: 1234@5678.com
22. # Disable minio as the default storage backend
23. minio:
24. enabled: false
25. # Configure Spinnaker to enable GCP services
26. halyard:
27. spinnakerVersion: 1.10.2
28. image:
29. tag: 1.12.0
30. additionalScripts:
31. create: true
32. data:
33. enable\_gcs\_artifacts.sh: |-
34. \$HAL\_COMMAND config artifact gcs account add gcs-$PROJECT --json-path /opt/gcs/key.json
35. \$HAL\_COMMAND config artifact gcs enable
36. enable\_pubsub\_triggers.sh: |-
37. \$HAL\_COMMAND config pubsub google enable
38. \$HAL\_COMMAND config pubsub google subscription add gcr-triggers \
39. --subscription-name gcr-triggers \
40. --json-path /opt/gcs/key.json \
41. --project $PROJECT \
42. --message-format GCR
43. 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.
2. ./helm install -n cd stable/spinnaker -f spinnaker-config.yaml \
3. --timeout 600 --version 1.1.6 --wait

Click *Check my progress* to verify the objective.

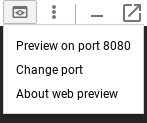
Deploying Spinnaker using Helm

Check my progress

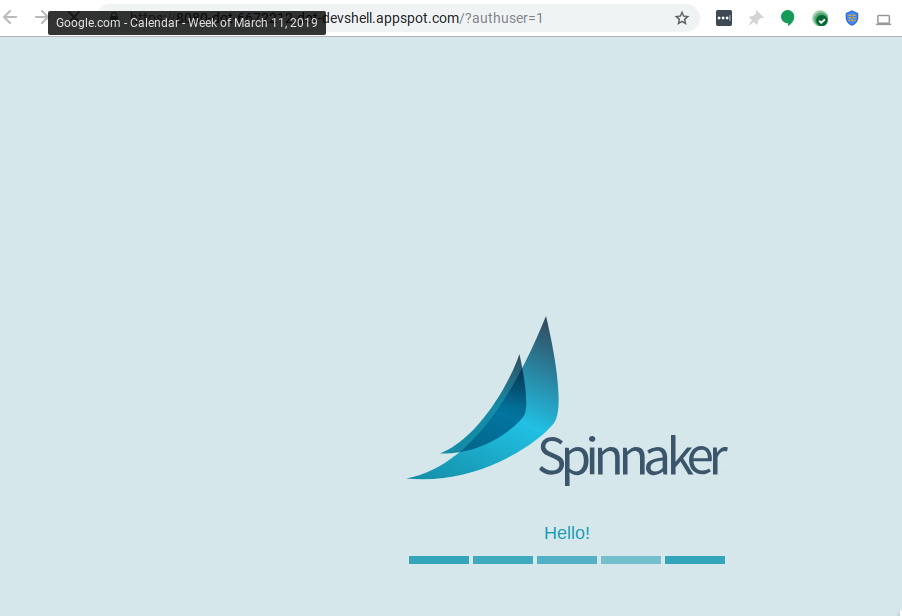
1. After the command completes, run the following command to set up port forwarding to the Spinnaker UI from Cloud Shell:
2. export DECK\_POD=$(kubectl get pods --namespace default -l "cluster=spin-deck" \
3. -o jsonpath="{.items[0].metadata.name}")
4. kubectl port-forward --namespace default $DECK\_POD 8080:9000 >> /dev/null &

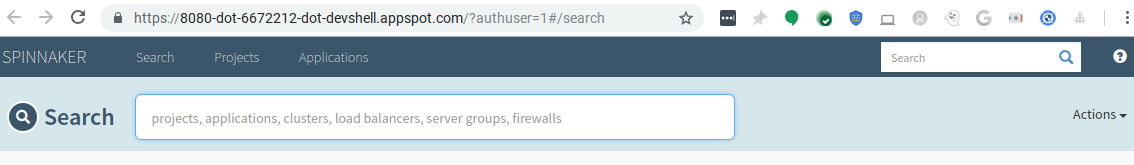
**Note:**This command can take several minutes to complete. Be sure to wait until you see that it has succeeded before proceeding.

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



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





**Building the Docker image**

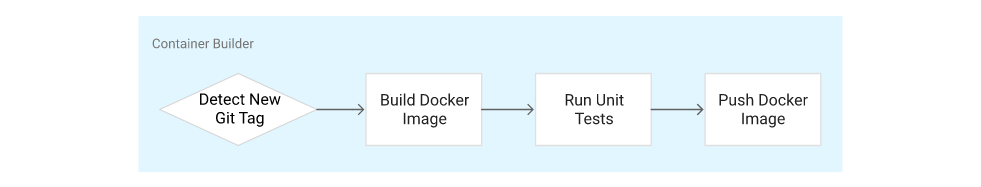
In this section, you configure Cloud Build to detect changes to your app source code, build a Docker image, and then push it to Container Registry.

Create your source code repository

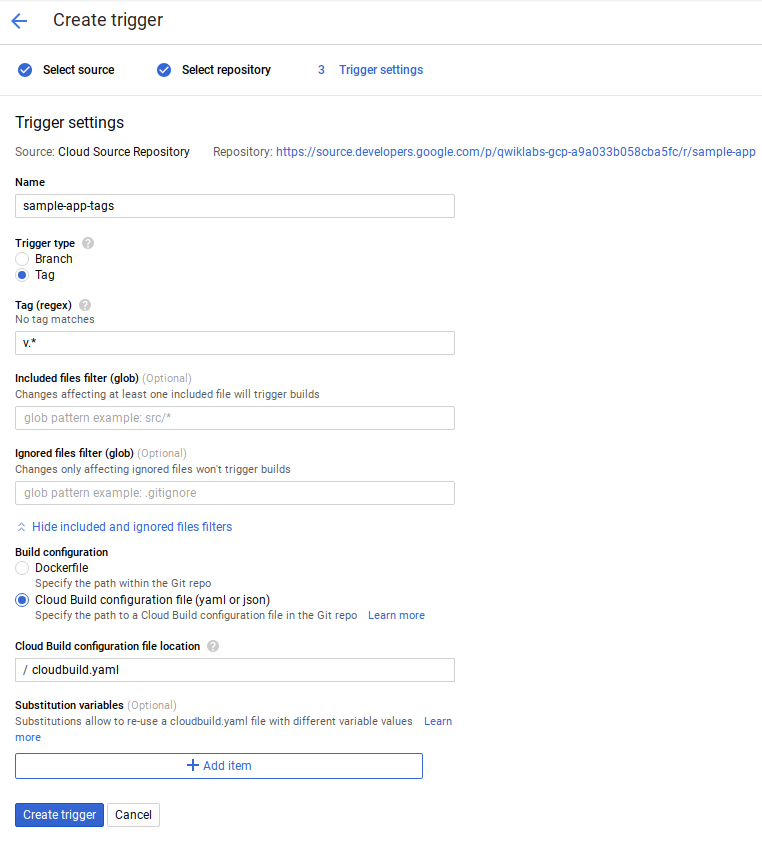
1. In Cloud Shell, download the sample source code:
2. wget https://gke-spinnaker.storage.googleapis.com/sample-app-v2.tgz
3. Unpack the source code:
4. tar xzfv sample-app-v2.tgz
5. Change directories to the source code:
6. cd sample-app
7. Set the username and email address for your Git commits in 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](https://git-scm.com/book/en/v2/Git-Basics-Tagging) 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**.



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

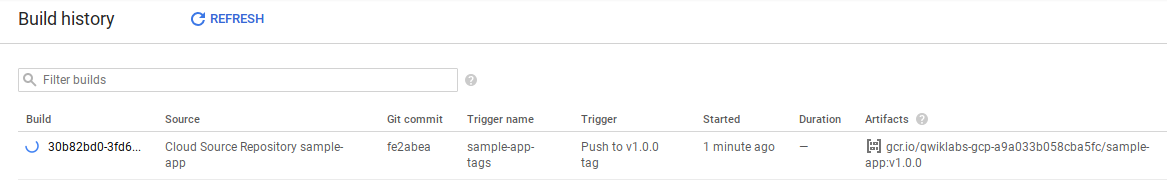
Spinnaker needs access to your Kubernetes manifests in order to deploy them to your clusters. This section creates a Cloud Storage bucket that will be populated with your manifests during the CI process in Cloud Build. After your manifests are in Cloud Storage, Spinnaker can download and apply them during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



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



Click *Check my progress* to verify the objective.

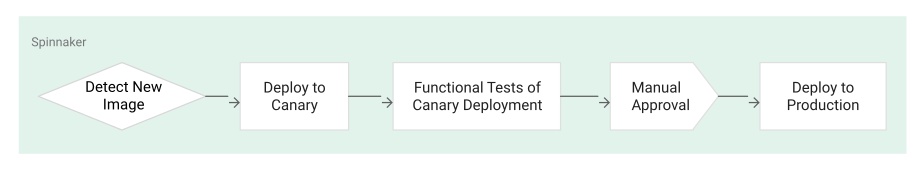
Building the Docker image

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](https://www.spinnaker.io/guides/spin/cli/) is a command-line utility for managing Spinnaker's applications and pipelines.

1. Download the 1.6.0 version of spin.
2. curl -LO https://storage.googleapis.com/spinnaker-artifacts/spin/1.6.0/linux/amd64/spin
3. Make spin executable.
4. 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.
2. ./spin application save --application-name sample \
3. --owner-email "$(gcloud config get-value core/account)" \
4. --cloud-providers kubernetes \
5. --gate-endpoint http://localhost:8080/gate

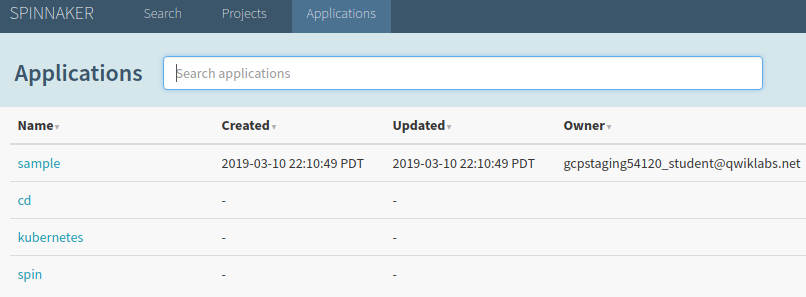
Next, you create the continuous delivery pipeline. In this tutorial, the pipeline is configured to detect when a Docker image with a tag prefixed with "v" has arrived in your Container Registry.

1. Open a **new tab** in Cloud Shell using **Add Cloud Shell session**.
2. Return to your sample-app source code directory and run the following command to upload an example pipeline to your Spinnaker instance:
3. export PROJECT=$(gcloud info --format='value(config.project)')
4. sed s/PROJECT/$PROJECT/g spinnaker/pipeline-deploy.json > pipeline.json
5. ./spin pipeline save --gate-endpoint http://localhost:8080/gate -f pipeline.json

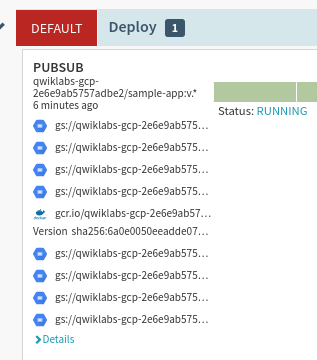
Manually Trigger and View your pipeline execution

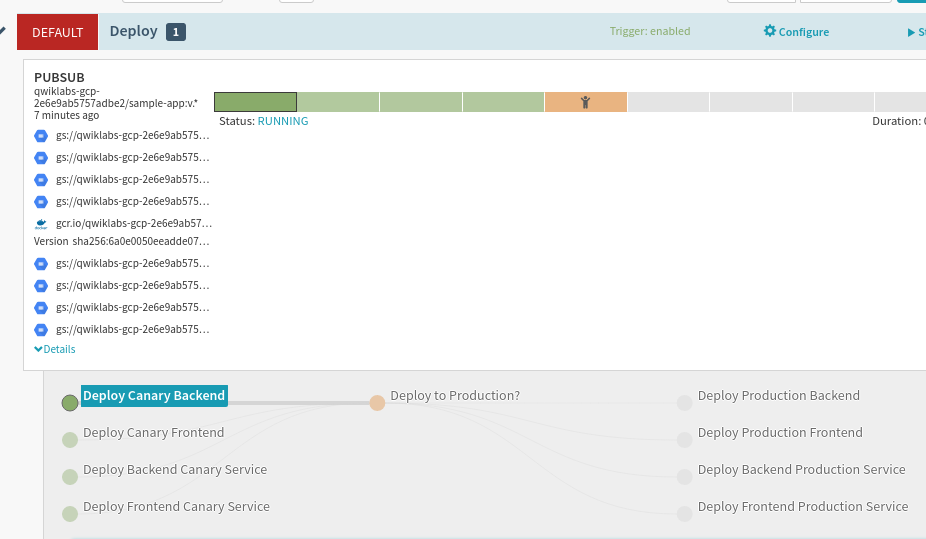
The configuration you just created uses notifications of newly tagged images being pushed to trigger a Spinnaker pipeline. In a previous step, you pushed a tag to the Cloud Source Repositories which triggered Cloud Build to build and push your image to Container Registry. You can now **manually trigger**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



1. Click **sample** to view your application deployment.
2. Click **Pipelines** at the top to view your applications pipeline status.
3. Click **Start Manual Execution** to trigger the pipeline this first time. Click **Run**.
4. 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.

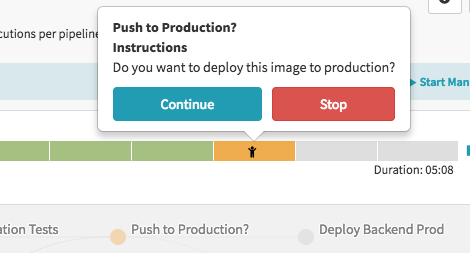




1. Click a stage to see details about it.

After **3 to 5 minutes** the integration test phase completes and the pipeline requires manual approval to continue the deployment.

1. 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

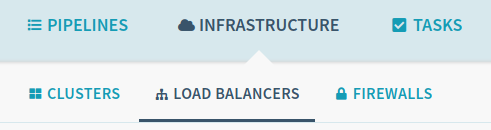
Check my progress

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

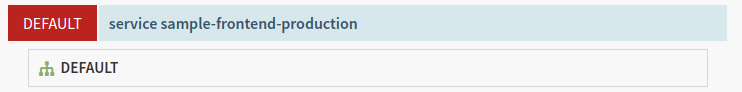
Deploy image to production

Check my progress

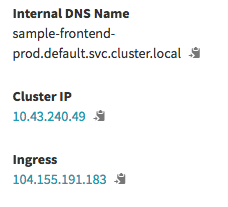
1. To view the app, select **Infrastructure** > **Load Balancers** in the top of the Spinnaker UI.



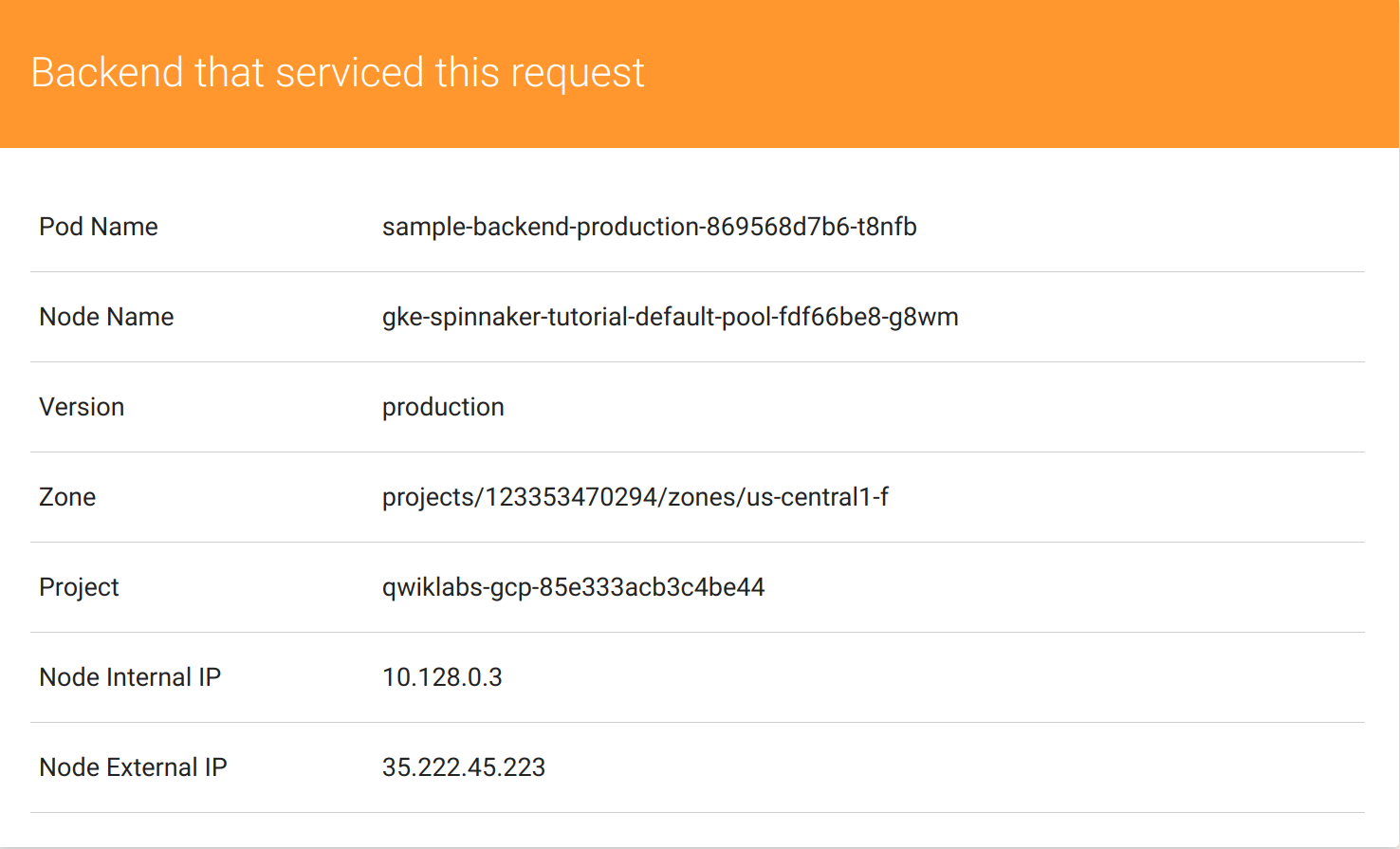
1. Scroll down the list of load balancers and click **Default**, under **service sample-frontend-production**.



1. 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.



1. 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.



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.

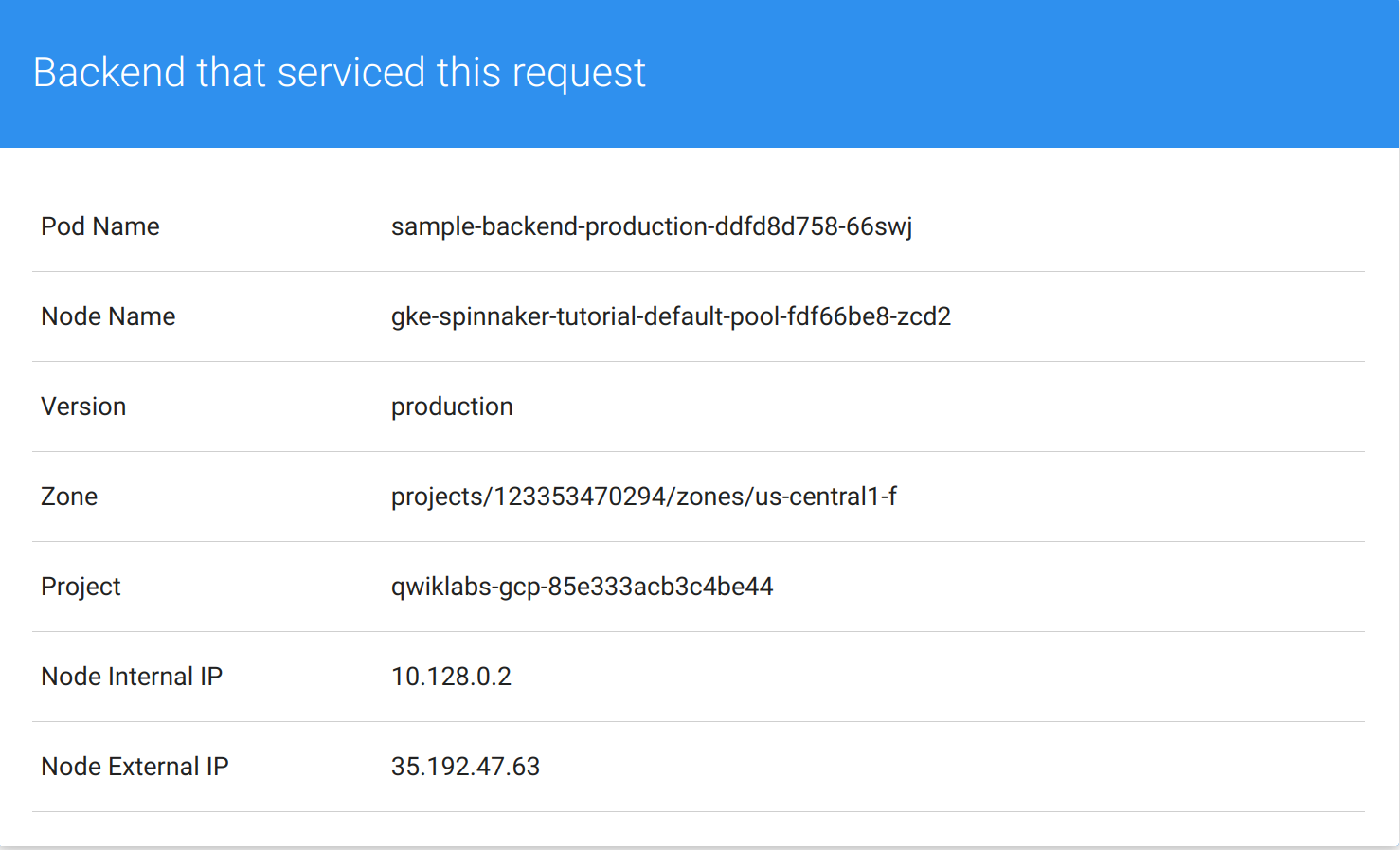
Click *Check my progress* to verify the objective.

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.



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

1. 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:
2. git revert v1.0.1
3. git tag v1.0.2
4. 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.