# Continuous Delivery Pipelines with Spinnaker and Kubernetes Engine

1 hour 30 minutes9 Credits

## **GSP114**



## **Overview**

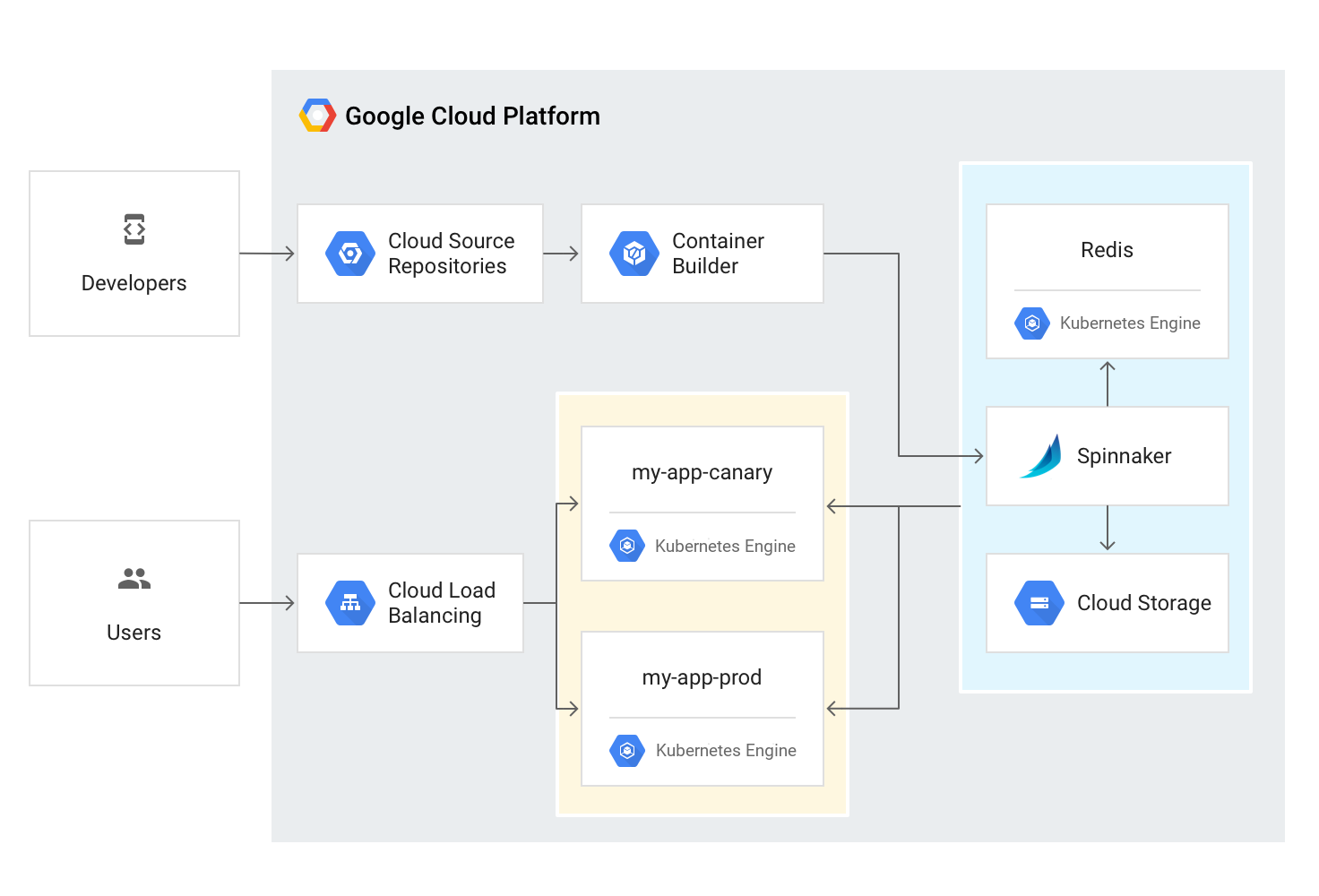
This hands-on lab shows you how to create a continuous delivery pipeline using Google Kubernetes Engine, Google Cloud Source Repositories, Google Cloud Container Builder, and Spinnaker. After you create a sample application, you configure these services to automatically build, test, and deploy it. When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

### Objectives

* Set up your environment by launching [Google Cloud Shell](https://cloud.google.com/shell/), creating a Kubernetes Engine cluster, and configuring your identity and user management scheme.
* Download a sample application, create a Git repository then upload it to a Google Cloud Source Repository.
* Deploy Spinnaker to Kubernetes Engine using [Helm](https://github.com/kubernetes/helm).
* Build your Docker image.
* Create triggers to create Docker images when your application changes.
* Configure a Spinnaker pipeline to reliably and continuously deploy your application to Kubernetes Engine.
* Deploy a code change, triggering the pipeline, and watch it roll out to production.

## **Pipeline architecture**

To continuously deliver application 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](https://martinfowler.com/bliki/CanaryRelease.html) releases proves unsatisfactory, your automated procedure must be able to quickly roll back the software changes.

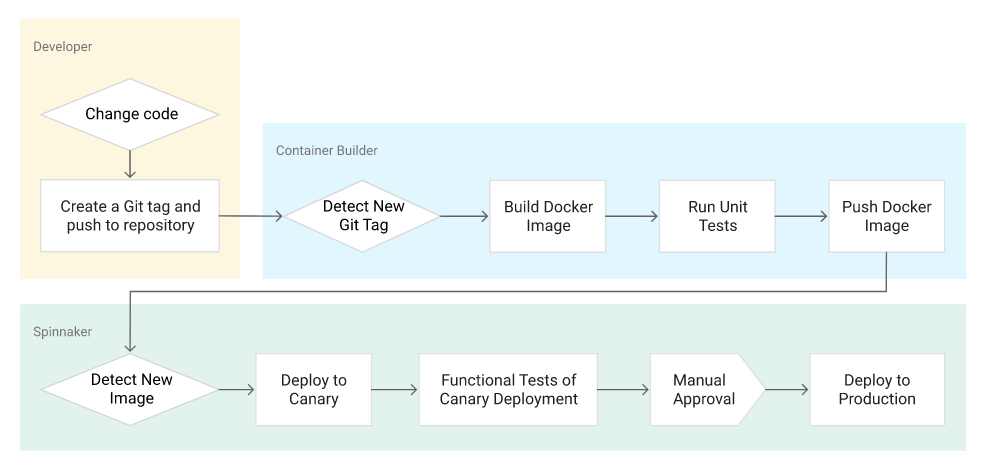


With Kubernetes Engine and Spinnaker you can create a robust continuous delivery flow that helps to ensure your software is shipped as quickly as it is developed and validated. Although rapid iteration is your end goal, you must first ensure that each application revision passes through a gamut of automated validations before becoming a candidate for production rollout. When a given change has been vetted through automation, you can also validate the application manually and conduct further pre-release testing.

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

### Application 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 Google Cloud resources will be made available to you.

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

#### **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 Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Chrome OS device, open an Incognito window to run this lab.

#### **How to start your lab and sign in to the Google Cloud Console**

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.
2. F,{37d9aeef-9c3f-4e1e-a67d-6b052d470132}{203},3.125,3.125
3. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.
4. 
5. ***Tip:*** Open the tabs in separate windows, side-by-side.
6. If you see the **Choose an account** page, click **Use Another Account**. 
7. In the **Sign in** page, paste the username that you copied from the left panel. Then copy and paste the password.
8. ***Important:*** You must use the credentials from the left panel. Do not use your Google Cloud Training credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).
9. 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 Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

### Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



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:



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

You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

(Output)

Credentialed accounts: - [<myaccount>@<mydomain>.com](mailto:<myaccount>@<mydomain>.com) (active)

(Example output)

Credentialed accounts: - [google1623327\_student@qwiklabs.net](mailto:google1623327_student@qwiklabs.net)

You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

(Output)

[core]project = <project\_ID>

(Example output)

[core]project = qwiklabs-gcp-44776a13dea667a6

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## **Set up your environment**

Configure the infrastructure and identities required for this lab. First you'll create a Kubernetes Engine cluster to deploy Spinnaker and the sample application.

1. Set the default compute zone:

gcloud config set compute/zone us-central1-f

Copied!

content\_copy

1. Create a Kubernetes Engine cluster using the Spinnaker tutorial sample application:

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

Copied!

content\_copy

Cluster creation takes between **five and ten minutes** to complete. Wait for your cluster to finish provisioning before proceeding.

When completed you see a report detailing the name, location, version, ip-address, machine-type, node version, number of nodes and status of the cluster that indicates the cluster is running.

### Configure identity and access management

Create a Cloud Identity 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.

Upload your startup script to a Cloud Storage bucket by following these steps:

1. Create the service account:

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

Copied!

content\_copy

1. 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)')

Copied!

content\_copy

export PROJECT=$(gcloud info --format='value(config.project)')

Copied!

content\_copy

1. Bind the storage.admin role to your service account:

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

Copied!

content\_copy

1. Download the service account key. In a later step, you will install Spinnaker and upload this key to Kubernetes Engine:

gcloud iam service-accounts keys create spinnaker-sa.json \ --iam-account $SA\_EMAIL

Copied!

content\_copy

(Output)

created key [12f224e036437704b91a571792462ca6fc4cd438] of type [json] as [spinnaker-sa.json] for [[spinnaker-account@qwiklabs-gcp-gcpd-f5e16da10e5d.iam.gserviceaccount.com](mailto:spinnaker-account@qwiklabs-gcp-gcpd-f5e16da10e5d.iam.gserviceaccount.com)]

## **Set up Cloud Pub/Sub to trigger Spinnaker pipelines**

1. Create the Cloud Pub/Sub topic for notifications from Container Registry.

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

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. 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)')

Copied!

content\_copy

gcloud beta pubsub subscriptions add-iam-policy-binding gcr-triggers \ --role roles/pubsub.subscriber --member serviceAccount:$SA\_EMAIL

Copied!

content\_copy

### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully setup the environment, you will see an assessment score.

Set up your environment

Check my progress

## **Deploying Spinnaker using Helm**

In this section you use [Helm](https://github.com/kubernetes/helm) to deploy Spinnaker from the [Charts](https://github.com/kubernetes/charts) repository. Helm is a package manager you can use to configure and deploy [Kubernetes applications](http://kubeapps.com/).

**Helm** is already installed in your **Cloud Shell**.

### Configure Helm

1. Grant Helm the cluster-admin role in your cluster:

kubectl create clusterrolebinding user-admin-binding \ --clusterrole=cluster-admin --user=$(gcloud config get-value account)

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. Add the stable charts deployments to Helm's usable repositories (includes Spinnaker):

helm repo add stable <https://charts.helm.sh/stable>helm repo update

Copied!

content\_copy

### Configure Spinnaker

1. Still in Cloud Shell, create a bucket for Spinnaker to store its pipeline configuration:

export PROJECT=$(gcloud info \ --format='value(config.project)')

Copied!

content\_copy

export BUCKET=$PROJECT-spinnaker-config

Copied!

content\_copy

gsutil mb -c regional -l us-central1 gs://$BUCKET

Copied!

content\_copy

1. 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-configcat > spinnaker-config.yaml <<EOFgcs: enabled: true bucket: $BUCKET project: $PROJECT jsonKey: '$SA\_JSON'dockerRegistries:- name: gcr address: <https://gcr.io> username: \_json\_key password: '$SA\_JSON' email: [1234@5678.com](mailto:1234@5678.com)# Disable minio as the default storage backendminio: enabled: false# Configure Spinnaker to enable GCP serviceshalyard: spinnakerVersion: 1.19.4 image: repository: us-docker.pkg.dev/spinnaker-community/docker/halyard tag: 1.32.0 pullSecrets: [] additionalScripts: create: true data: enable\_gcs\_artifacts.sh: |- \$HAL\_COMMAND config artifact gcs account add gcs-$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 GCREOF

Copied!

content\_copy

### Deploy the Spinnaker chart

1. Use the Helm command-line interface to deploy the chart with your configuration set:

helm install -n default cd stable/spinnaker -f spinnaker-config.yaml \ --version 2.0.0-rc9 --timeout 10m0s --wait

Copied!

content\_copy

The installation typically takes **5-8 minutes** to complete.

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

export DECK\_POD=$(kubectl get pods --namespace default -l "cluster=spin-deck" \ -o jsonpath="{.items[0].metadata.name}")

Copied!

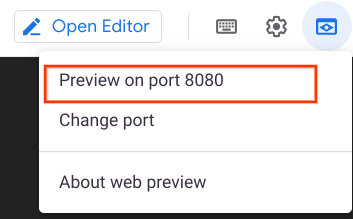
content\_copy

kubectl port-forward --namespace default $DECK\_POD 8080:9000 >> /dev/null &

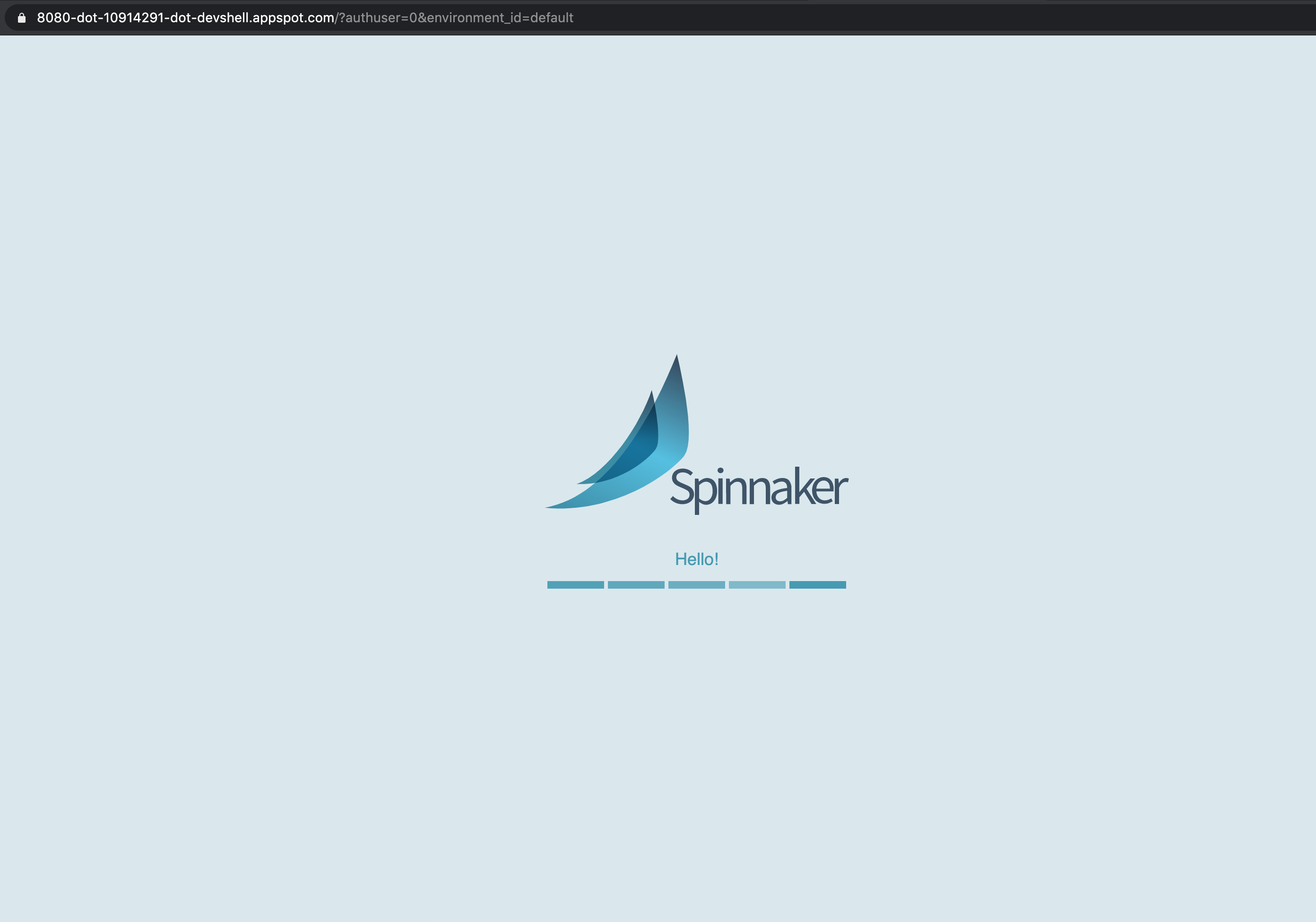
Copied!

content\_copy

1. To open the Spinnaker user interface, click the **Web Preview** icon at the top of the Cloud Shell window and select **Preview on port 8080**.



The welcome screen opens, followed by the Spinnaker user interface:



Leave this tab open, this is where you'll access the Spinnaker UI.

### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully deployed the Spinnaker chart using Kubernetes Helm, you will see an assessment score.

Deploy the Spinnaker chart using Kubernetes Helm

Check my progress

## **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 tab, download the sample application source code:

gsutil -m cp -r gs://spls/gsp114/sample-app.tar .

Copied!

content\_copy

1. Unpack the source code:

mkdir sample-apptar xvf sample-app.tar -C ./sample-app

Copied!

content\_copy

1. Change directories to the source code:

cd sample-app

Copied!

content\_copy

1. Set the username and email address for your Git commits in this repository. Replace [USERNAME] with a username you create:

git config --global user.email "$(gcloud config get-value core/account)"

Copied!

content\_copy

git config --global user.name "[USERNAME]"

Copied!

content\_copy

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

git init

Copied!

content\_copy

git add .

Copied!

content\_copy

git commit -m "Initial commit"

Copied!

content\_copy

1. Create a repository to host your code:

gcloud source repos create sample-app

Copied!

content\_copy

Disregard the "you may be billed for this repository" message.

git config credential.helper gcloud.sh

Copied!

content\_copy

1. Add your newly created repository as remote:

export PROJECT=$(gcloud info --format='value(config.project)')

Copied!

content\_copy

git remote add origin <https://source.developers.google.com/p/$PROJECT/r/sample-app>

Copied!

content\_copy

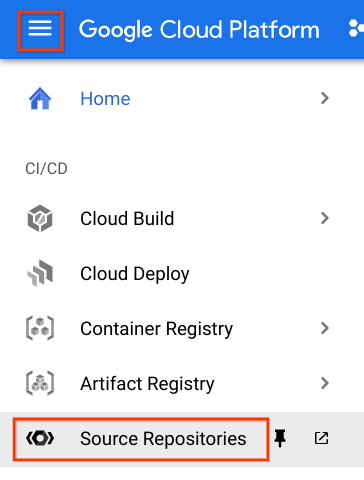
1. Push your code to the new repository's master branch:

git push origin master

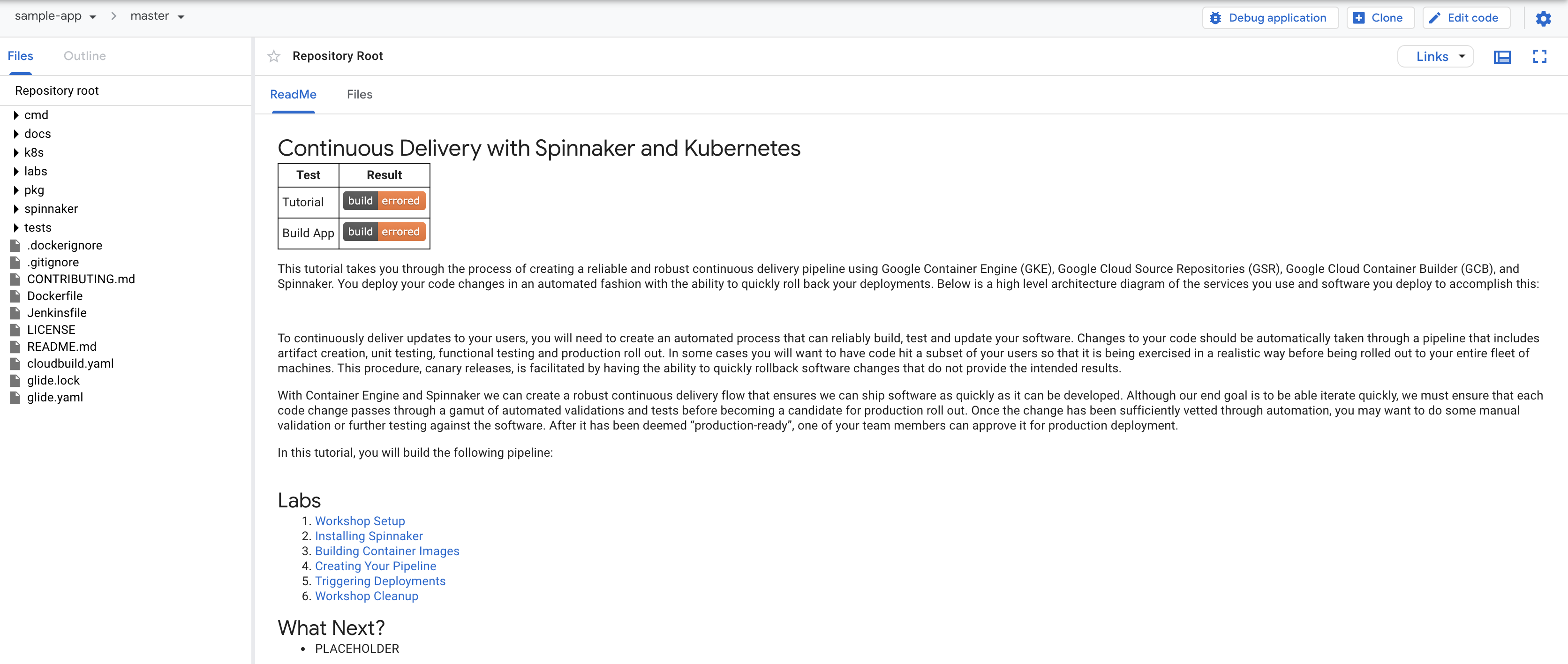
Copied!

content\_copy

1. Check that you can see your source code in the Console by clicking **Navigation Menu** > **Source Repositories**.

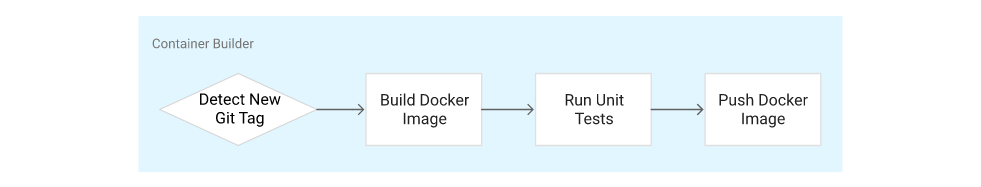


1. Click **sample-app**.

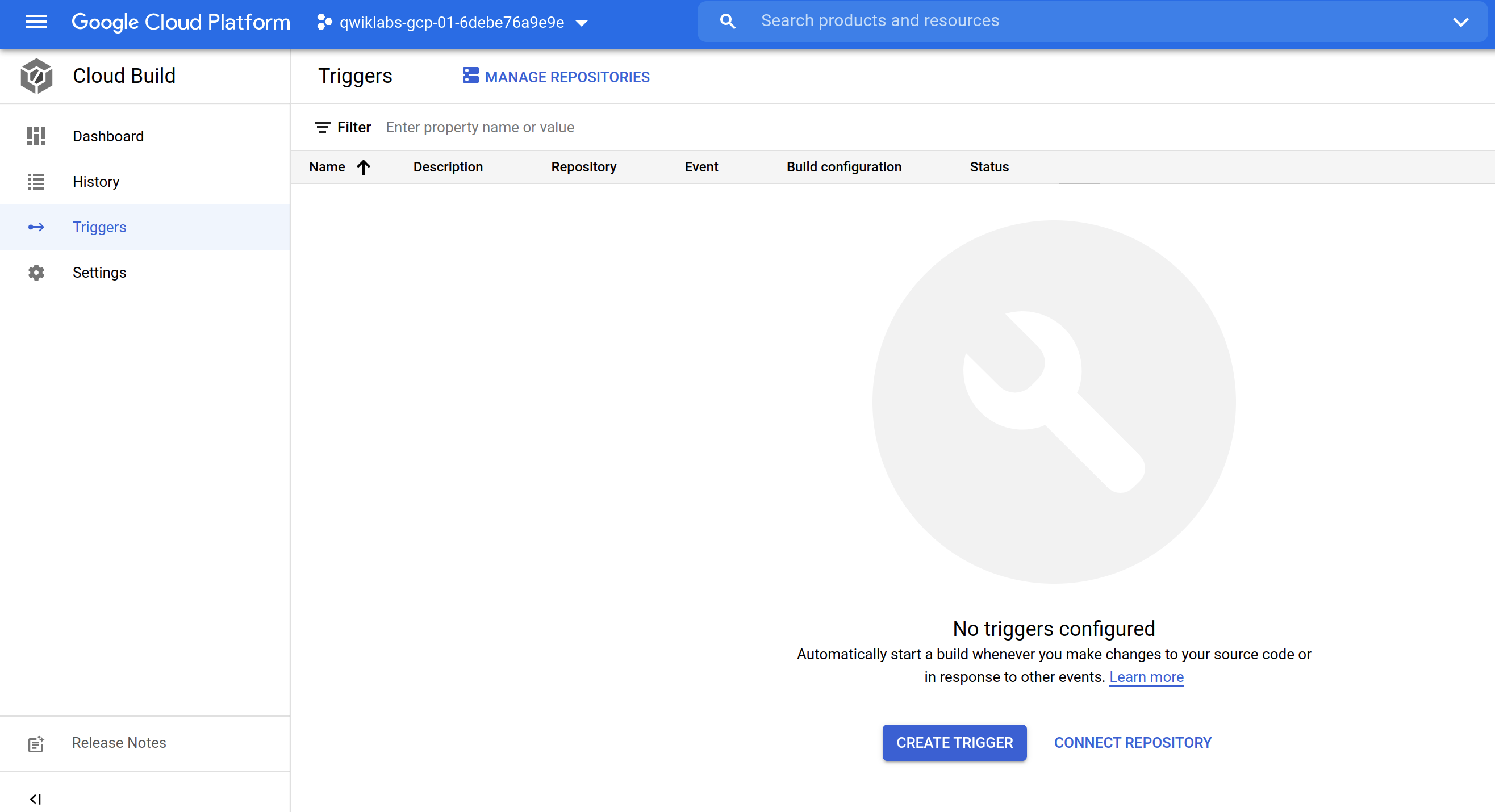


### Configure your build triggers

Configure Container Builder 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. Container Builder automatically checks out your source code, builds the Docker image from the Dockerfile in your repository, and pushes that image to Google Cloud Container Registry.



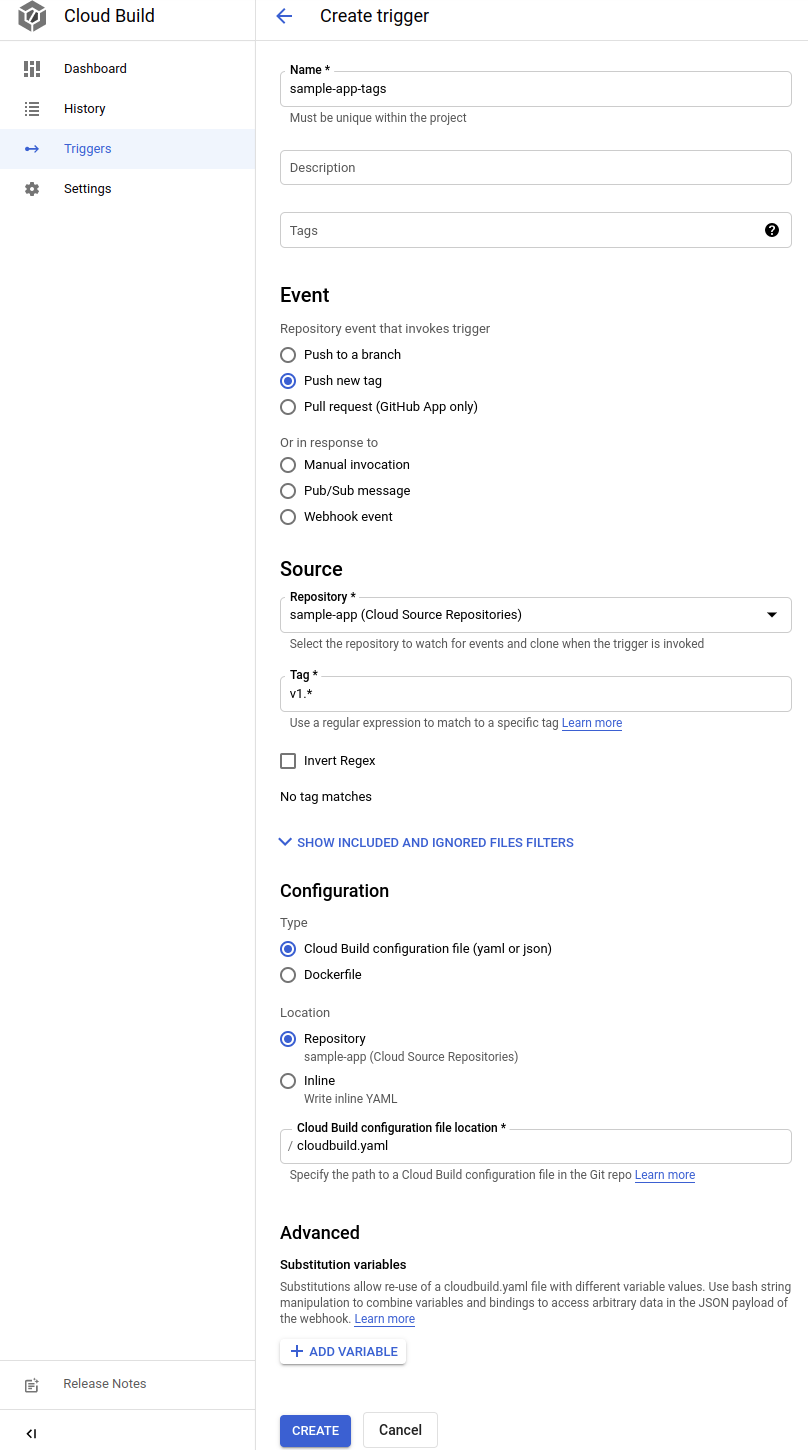
1. In the Cloud Platform Console, click **Navigation menu** > **Cloud Build** > **Triggers**.
2. Click **Create trigger**.



1. Set the following trigger settings:

* **Name**: sample-app-tags
* **Event**: Push new tag
* Select your newly created sample-app repository.
* **Tag**: v1.\*
* **Configuration**: Cloud Build configuration file (yaml or json)
* **Cloud Build configuration file location**: /cloudbuild.yaml

1. Click **CREATE**.



From now on, whenever you push a Git tag prefixed with the letter "v" to your source code repository, Container Builder automatically builds and pushes your application 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:

export PROJECT=$(gcloud info --format='value(config.project)')

Copied!

content\_copy

gsutil mb -l us-central1 gs://$PROJECT-kubernetes-manifests

Copied!

content\_copy

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

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

Copied!

content\_copy

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

sed -i s/PROJECT/$PROJECT/g k8s/deployments/\*

Copied!

content\_copy

1. Commit the changes to the repository:

git commit -a -m "Set project ID"

Copied!

content\_copy

### Build your image

Push your first image using the following steps:

1. In Cloud Shell, still in the sample-app directory, create a Git tag:

git tag v1.0.0

Copied!

content\_copy

1. Push the tag:

git push --tags

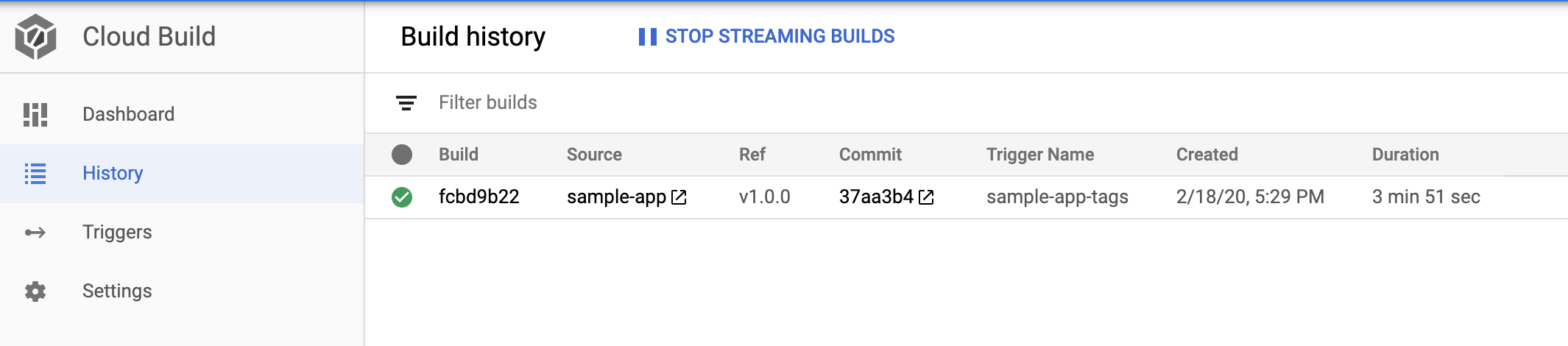
Copied!

content\_copy

(output)

To <https://source.developers.google.com/p/qwiklabs-gcp-ddf2925f84de0b16/r/sample-app>\* [new tag] v1.0.0 -> v1.0.0

1. Go to the Cloud Console. Still in Cloud Build, click **History** in the left pane to check that the build has been triggered. If not, verify that the trigger was configured properly in the previous section.



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

**Note:** If the Build fails, then click on the Build ID to open Build details page and then click **RETRY**.

### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully build the Docker image, you will see an assessment score.

Build 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://spinnaker.io/docs/guides/spin/) is a command-line utility for managing Spinnaker's applications and pipelines.

1. Download the 1.14.0 version of spin:

curl -LO <https://storage.googleapis.com/spinnaker-artifacts/spin/1.14.0/linux/amd64/spin>

Copied!

content\_copy

1. Make spin executable:

chmod +x spin

Copied!

content\_copy

### 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>

Copied!

content\_copy

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. From your sample-app source code directory, 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

Copied!

content\_copy

### Manually Trigger and View your pipeline execution

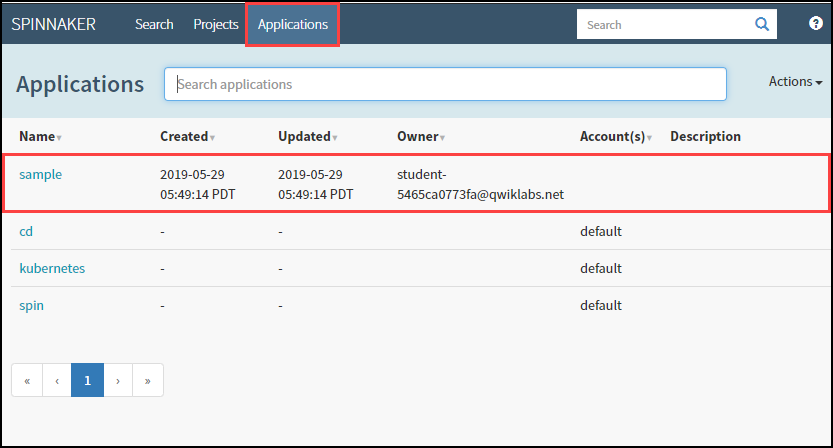
The configuration you just created uses notifications of newly tagged images being pushed to trigger a Spinnaker pipeline. In a previous step, you pushed a tag to the Cloud Source Repositories which triggered Cloud Build to build and push your image to Container Registry. To verify the pipeline, manually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

1. Switch to your browser tab displaying your Spinnaker UI.

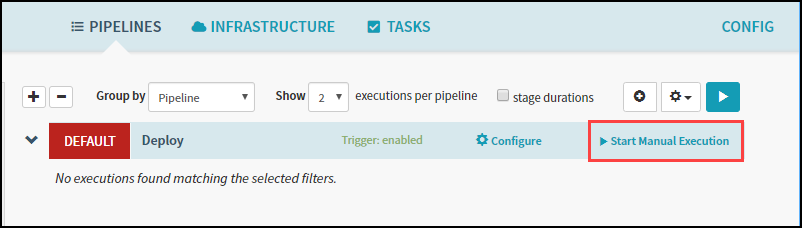
If you are unable to find it, you can get to this tab again by selecting **Web Preview** > **Preview on Port 8080** in your Cloud Shell window.

1. In the Spinnaker UI, click **Applications** at the top of the screen to see your list of managed applications.

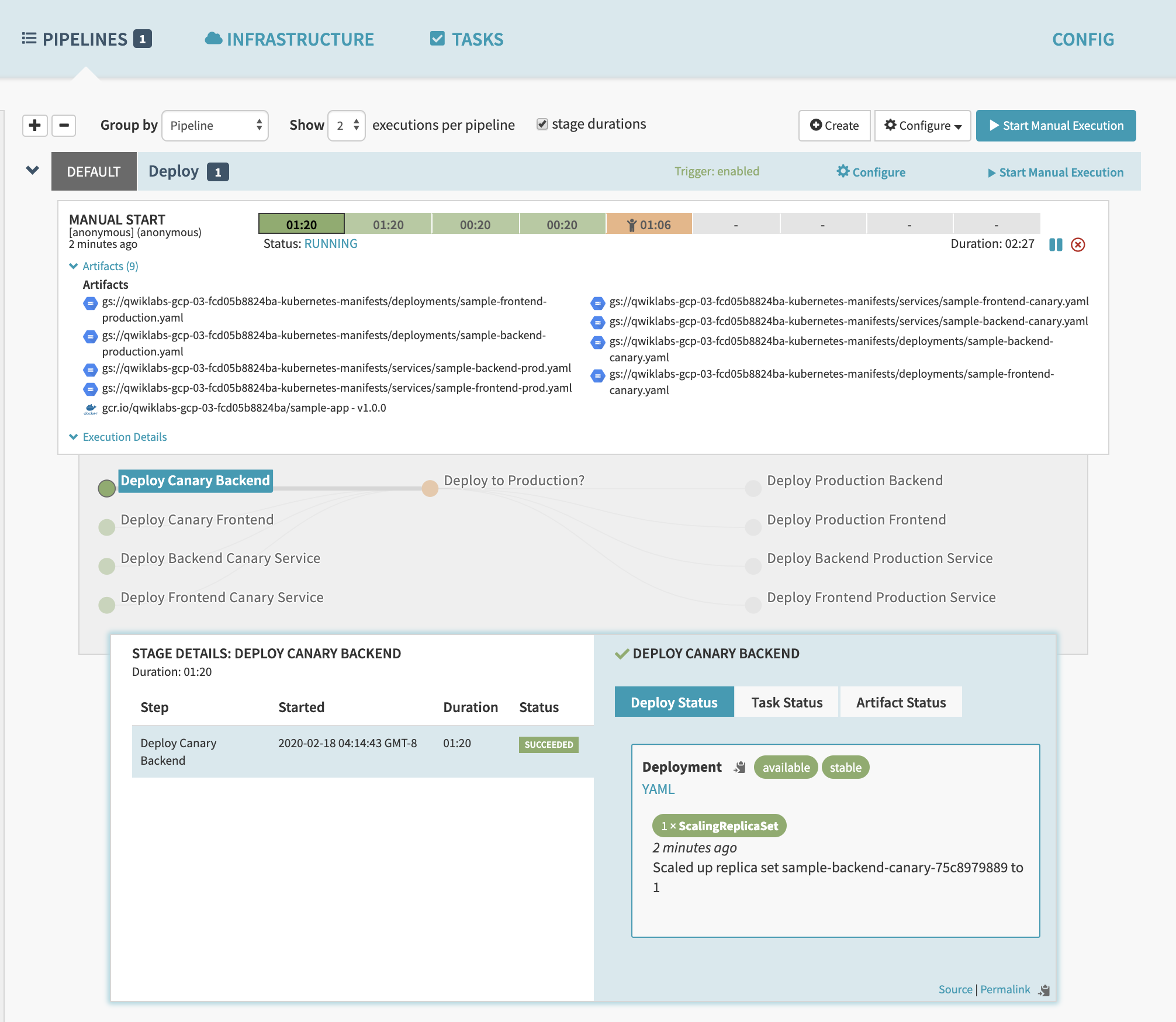
**sample** is your application. If you don't see **sample**, try refreshing the Spinnaker Applications tab.



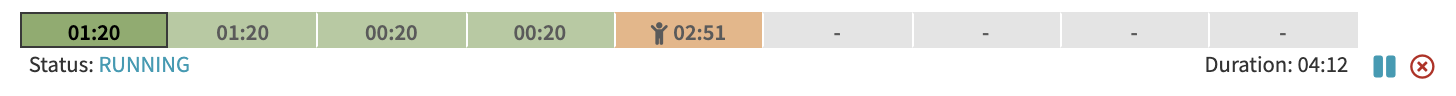
1. Click **sample** to view your application deployment.
2. Click **Pipelines** at the top to view your applications pipeline status.
3. Click **Start Manual Execution** and then click **Run** to trigger the pipeline this first time.



1. Click **Execution Details** to see more information about the pipeline's progress.



The progress bar shows the status of the deployment pipeline and its steps.

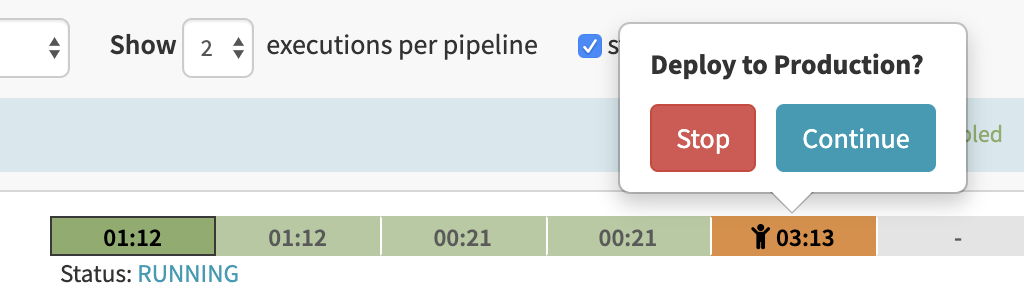


Steps in blue are currently running, green ones have completed successfully, and red ones have failed.

1. Click a stage to see details about it.

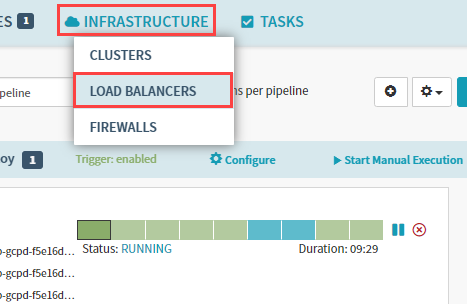
After **3 to 5 minutes** the integration test phase completes and the pipeline requires manual approval to continue the deployment.

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.

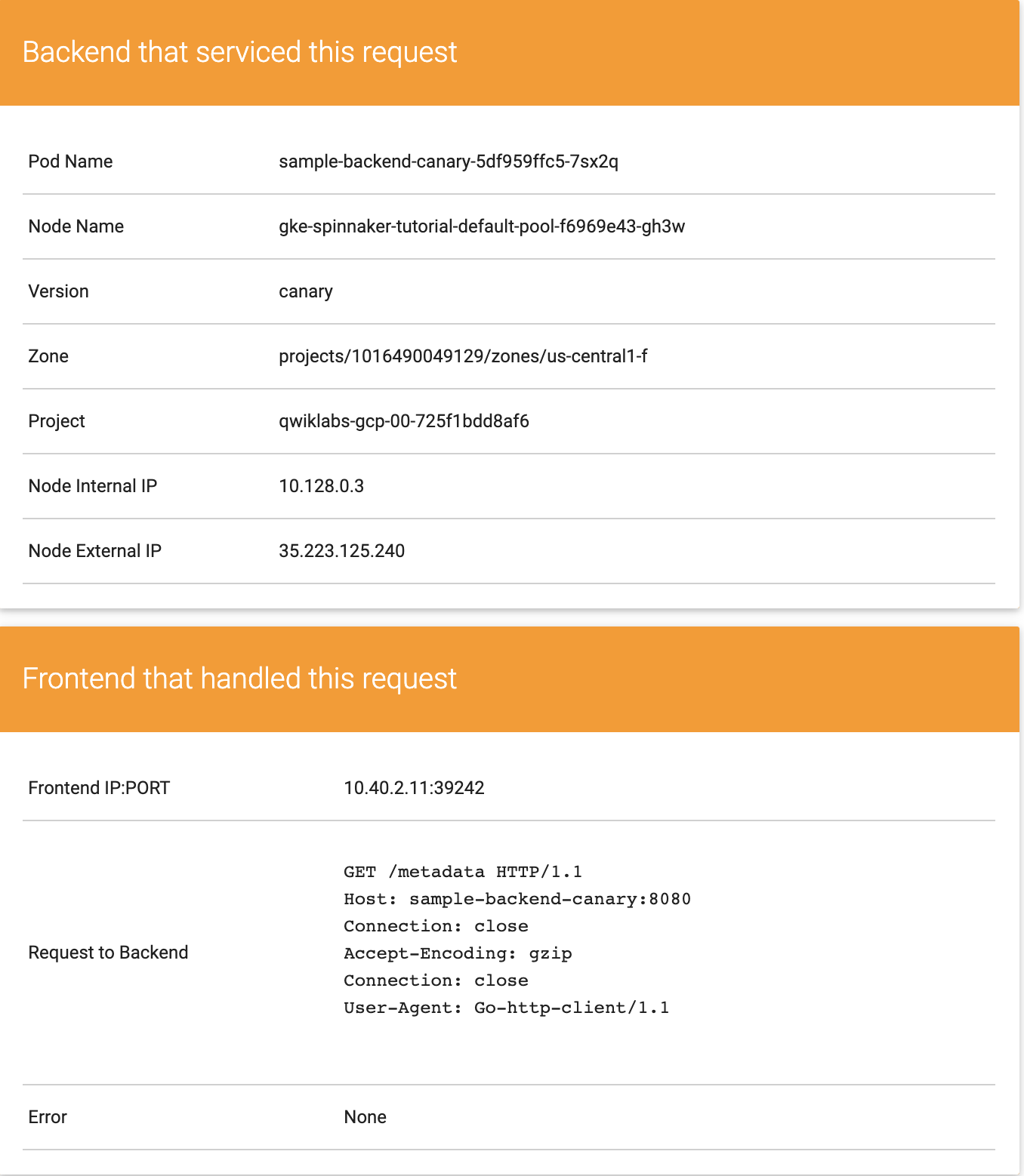
1. To view the app, select **Infrastructure** > **Load Balancers** in the top of the Spinnaker UI.



1. Scroll down the list of load balancers and click Default, under service sample-frontend-production. You will see details for your load balancer appear on the right side of the page. If you do not, you may need to refresh your browser.
2. 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 a new browser tab to view the application. You might see the canary version displayed, but if you refresh you will also see the production version.



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

### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully created service load balancers, you will see an assessment score.

Create service load balancers

Check my progress

### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully deployed an image to production, you will see an assessment score.

Deploy an image to production

Check my progress

## **Triggering your pipeline from code changes**

Now 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 Container Builder 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:

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

Copied!

content\_copy

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

git commit -a -m "Change color to blue"

Copied!

content\_copy

git tag v1.0.1

Copied!

content\_copy

git push --tags

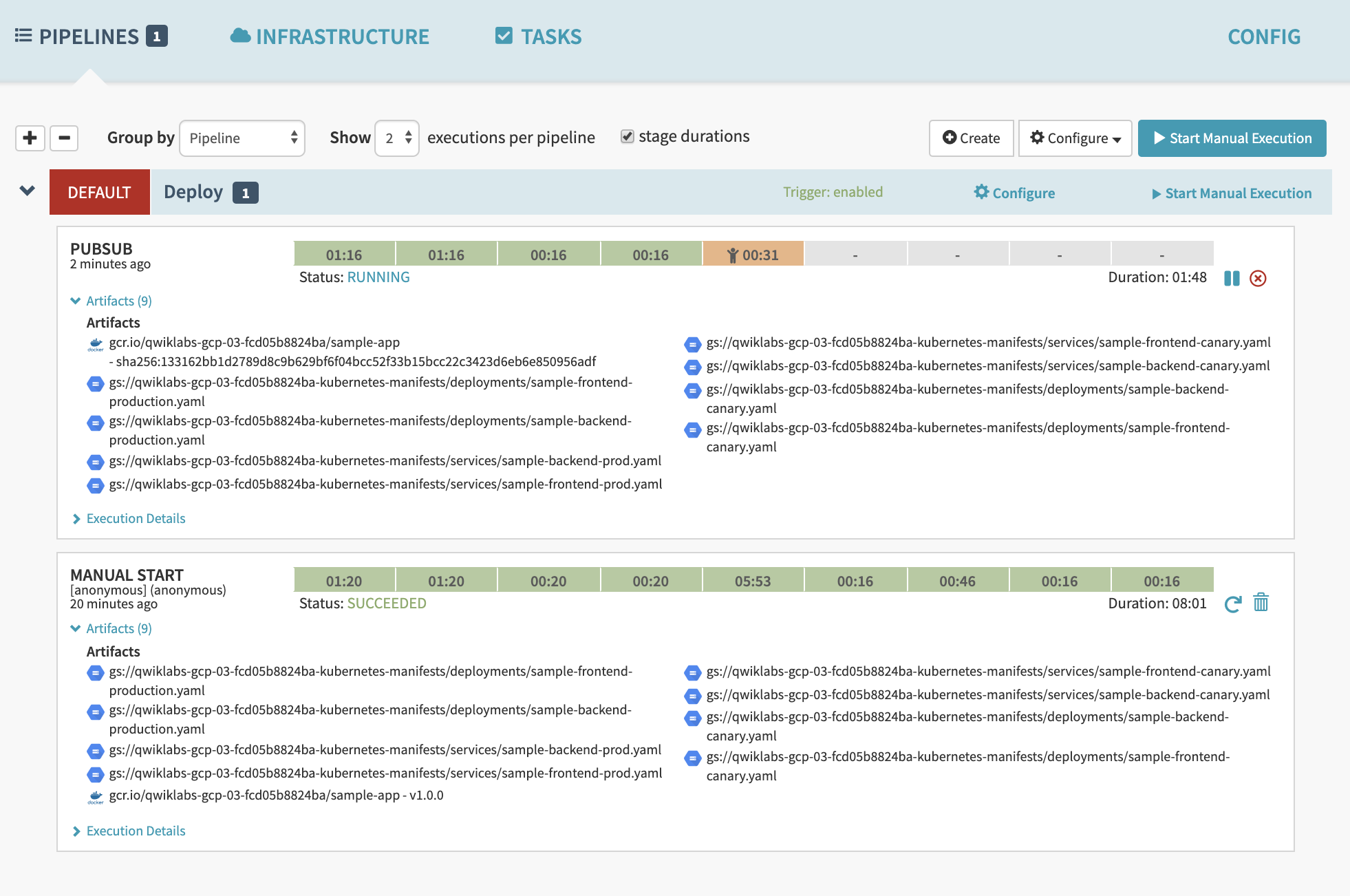
Copied!

content\_copy

1. In the Console, in **Cloud Build** > **History**, wait a couple of minutes for the new build to appear. You may need to refresh your page. Wait for the new build to complete, before going to the next step.

**Note:** If the Build fails, please click on Build ID and then click **RETRY**.

1. Return to the Spinnaker UI and click **Pipelines** to watch the pipeline start to deploy the image. The automatically triggered pipeline will take a few minutes to appear. You may need to refresh your page.



### Test Completed Task

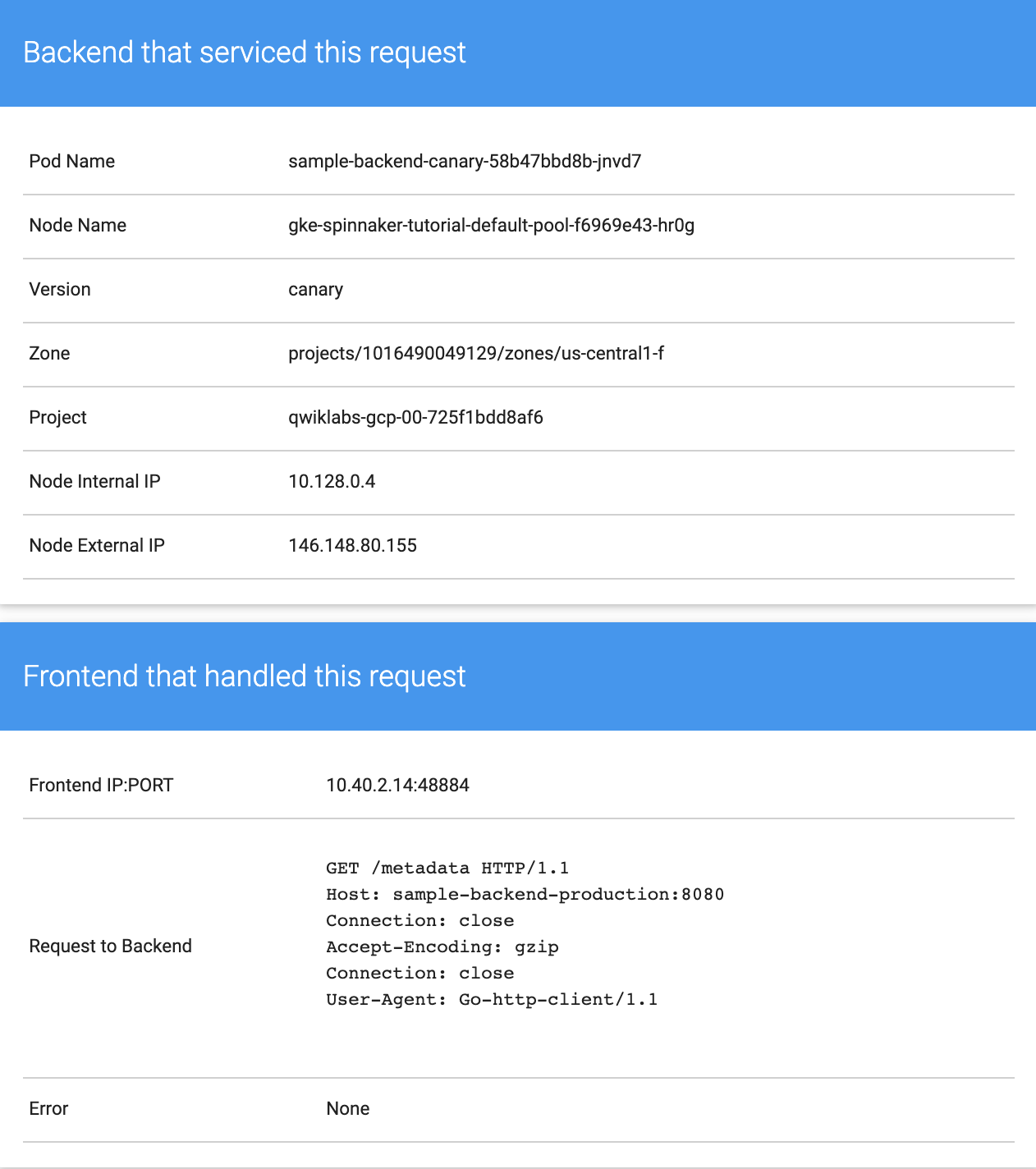
Click **Check my progress** to verify your performed task. If you have successfully triggered pipeline from code changes, you will see an assessment score.

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. 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 canary.



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:

git revert v1.0.1

Copied!

content\_copy

Press **CTRL+O**, **ENTER**, **CTRL+X**.

git tag v1.0.2

Copied!

content\_copy

git push --tags

Copied!

content\_copy

1. When the build and then the pipeline completes, verify the roll back by clicking **Infrastructure** > **Load Balancers**, then click the **service sample-frontend-production** **Default** and copy the Ingress IP address into a new tab.

Now your app is back to orange and you can see the production version number.

