Cloud Run Canary Deployments

experiment Lab schedule 1 hour 30 minutes

universal_currency_alt 5 Credits show_chart Intermediate

(/focuses/52827/reviews?parent=catalog)

info This lab may incorporate AI tools to support your learning.

Lab instructions and tasks expand_less

GSP1078

Google Cloud Self-Paced Labs

Overview

In this lab you will learn how to implement a deployment pipeline for Cloud Run that executes a progression of code from developer branches to production with automated canary testing and percentage based traffic management. It is intended for developers and DevOps engineers who are responsible for creating and managing CI/CD pipelines to Cloud Run.

Many organizations use robust release pipelines to move code into production. Cloud Run provides unique traffic management capabilities that let you implement advanced release management techniques with little effort.

Objectives

In this lab, you will learn how to:

- Create your Cloud Run service.
- Enable developer branch.
- Implement canary testing.
- Rollout safely to production.

Prerequisites

This lab assumes that you have a basic understanding of Git, Cloud Run, and CI/CD pipeline concepts.

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.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

> **Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

> **Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

How to start your lab and sign in to the Google Cloud console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is the **Lab Details** panel with the following:

    - The **Open Google Cloud console** button

- Time remaining
- The temporary credentials that you must use for this lab
- Other information, if needed, to step through this lab

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

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

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

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

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

   ```
   "Username"
   ```

   content_c

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

4. Click **Next**.

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

   ```
   "Password"
   ```

   content_c

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

6. Click **Next**.

   > **Important:** You must use the credentials the lab provides you. Do not use

your Google Cloud account credentials.

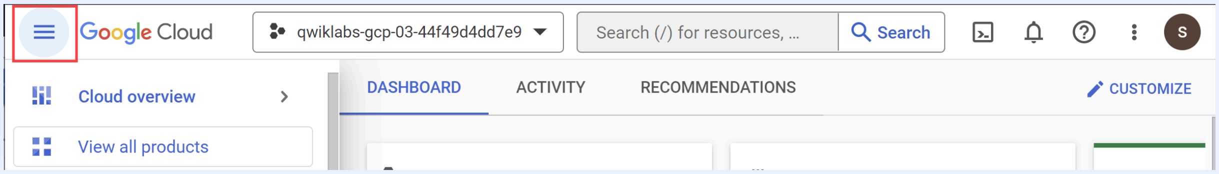
Note: Using your own Google Cloud account for this lab may incur extra charges.

7. Click through the subsequent pages:

    - Accept the terms and conditions.
    - Do not add recovery options or two-factor authentication (because this is a temporary account).
    - Do not sign up for free trials.

After a few moments, the Google Cloud console opens in this tab.

Note: To view a menu with a list of Google Cloud products and services, click the **Navigation menu** at the top-left.



Activate Cloud Shell

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

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

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

```
Your Cloud Platform project in this session is set to "PROJECT_ID"
```

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

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

```
gcloud auth list
```

content_c

3. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: "ACCOUNT"

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

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

```
gcloud config list project
```

content_c

Output:

```
[core]
project = "PROJECT_ID"
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide (https://cloud.google.com/sdk/gcloud).

Task 1. Preparing your environment

1. In Cloud Shell, create environment variables to use in this lab:

content_c

```
export PROJECT_ID=$(gcloud config get-value project)
export PROJECT_NUMBER=$(gcloud projects describe $PROJECT_ID --form
export REGION=
gcloud config set compute/region $REGION
```

2. Enable the following APIs with the code below:

- Cloud Resource Manager
- GKE
- Cloud Source Repositories
- Cloud Build
- Container Registry
- Cloud Run

content_c

```
gcloud services enable \
cloudresourcemanager.googleapis.com \
container.googleapis.com \
sourcerepo.googleapis.com \
cloudbuild.googleapis.com \
containerregistry.googleapis.com \
run.googleapis.com
```

3. Grant the Cloud Run Admin role (roles/run.admin) to the Cloud Build service account:

content_c

```
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member=serviceAccount:$PROJECT_NUMBER@cloudbuild.gserviceaccount.
--role=roles/run.admin
```

4. Grant the IAM Service Account User role (roles/iam.serviceAccountUser) to the Cloud Build service account for the Cloud Run runtime service account:

```
gcloud iam service-accounts add-iam-policy-binding \
$PROJECT_NUMBER-compute@developer.gserviceaccount.com \
--
member=serviceAccount:$PROJECT_NUMBER@cloudbuild.gserviceaccount.cc
\
--role=roles/iam.serviceAccountUser
```

5. If you haven't used Git in Cloud Shell previously, set the `user.name` and `user.email` values that you want to use (it's not necessary to have an existing account on GitHub):

```
git config --global user.email "[YOUR_EMAIL_ADDRESS]"
git config --global user.name "[YOUR_USERNAME]"
```

6. Clone and prepare the sample repository:

```
git clone https://github.com/GoogleCloudPlatform/software-
delivery-workshop --branch cloudrun-progression-csr cloudrun-
progression
cd cloudrun-progression/labs/cloudrun-progression
rm -rf ../../.git
```

7. Using **nano**, **vi** or any editor replace the `REGION` in `branch-cloudbuild.yaml` , `master-cloudbuild.yaml` and `tag-cloudbuild.yaml` files with the pre-populated `REGION` in **Step 1**.

8. Replace the placeholder values in the sample repository with your PROJECT_ID:

```
sed "s/PROJECT/${PROJECT_ID}/g" branch-trigger.json-tmpl >
branch-trigger.json
```

```
sed "s/PROJECT/${PROJECT_ID}/g" master-trigger.json-tmpl >
master-trigger.json
sed "s/PROJECT/${PROJECT_ID}/g" tag-trigger.json-tmpl > tag-
trigger.json
```

9. Store the code from the sample repository in Google Source Repository:

```
gcloud source repos create cloudrun-progression
git init
git config credential.helper gcloud.sh
git remote add gcp
https://source.developers.google.com/p/$PROJECT_ID/r/cloudrun-
progression
git branch -m master
git add . && git commit -m "initial commit"
git push gcp master
```

content_c

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

○ Enable the required services, grant the desired roles and store the code in source repository

Check my progress

Task 2. Creating your Cloud Run service

In this section, you build and deploy the initial production application that you use throughout this lab.

1. In Cloud Shell, build and deploy the application, including a service that requires authentication. To make a public service use the `--allow-unauthenticated` flag as

explained in the Cloud Run documentation (https://cloud.google.com/run/docs/authenticating/public).

```
gcloud builds submit --tag gcr.io/$PROJECT_ID/hello-cloudrun
gcloud run deploy hello-cloudrun \
--image gcr.io/$PROJECT_ID/hello-cloudrun \
--platform managed \
--region $REGION \
--tag=prod -q
```

content_c

The output looks similar to the following:

```
Deploying container to Cloud Run service [hello-cloudrun] in project [sdw-r
✓ Deploying new service... Done.
✓ Creating Revision...
✓ Routing traffic...
Done.
Service [hello-cloudrun] revision [hello-cloudrun-00001-tar] has been deplo
Service URL: https://hello-cloudrun-apwaaxltma-uc.a.run.app
The revision can be reached directly at https://prod---hello-cloudrun-apwaa
```

The output includes the service URL and a unique URL for the revision. Your values will differ slightly from what's indicated here.

2. After the deployment is complete, view the newly deployed service by selecting **Cloud Run** from the Cloud Console menu, choosing the **hello-cloudrun service**, and selecting the **Revisions** page.

3. In Cloud Shell, view the authenticated service response:

```
PROD_URL=$(gcloud run services describe hello-cloudrun --
platform managed --region $REGION --format=json | jq --raw-
output ".status.url")
echo $PROD_URL
```

content_c

```
curl -H "Authorization: Bearer $(gcloud auth print-identity-
token)" $PROD_URL
```

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

○ Create the CloudRun service and view the authenticated service response

Check my progress

Task 3. Enabling Dynamic Developer Deployments

In this section, you enable developers with a unique URL for development branches in Git. Each branch is represented by a URL identified by the branch name. Commits to the branch trigger a deployment, and the updates are accessible at that same URL.

1. In Cloud Shell, set up the trigger:

```
gcloud beta builds triggers create cloud-source-repositories --
trigger-config branch-trigger.json
```

content_c

2. To review the trigger, select **Cloud Build** from the Cloud Console menu and select **Triggers**.

3. In Cloud Shell, create a new branch:

```
git checkout -b new-feature-1
```

content_c

4. Open the sample application using your favorite editor or using the Cloud Shell IDE:

```
edit app.py
```

content_c

5. You can now browse or modify the code with the editor. In the sample application (**~/cloudrun-progression/labs/cloudrun-progression/app.py**), modify line 24 to indicate `v1.1` instead of `v1.0`:

```
@app.route('/')
def hello_world():
return 'Hello World v1.1'
```

content_c

6. To return to your terminal, click **Open Terminal**.

7. In Cloud Shell, commit the change and push to the remote repository:

```
git add . && git commit -m "updated" && git push gcp new-
feature-1
```

content_c

8. To review the build in progress, go back to the **Cloud Build** page and view the current build running on your new branch.

9. After the build completes, to review the revision, go to **Cloud Run** from the Cloud Console menu, choose the **hello-cloudrun service**, and select the **Revisions** page.

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

○ Set up the branch trigger and update the sample application

Check my progress

10. In Cloud Shell, get the unique URL for this branch:

```
BRANCH_URL=$(gcloud run services describe hello-cloudrun --
platform managed --region $REGION --format=json | jq --raw-
output ".status.traffic[] | select (.tag==\"new-feature-
1\")|.url")
echo $BRANCH_URL
```

content_c

11. Access the authenticated URL:

```
curl -H "Authorization: Bearer $(gcloud auth print-identity-
token)" $BRANCH_URL
```

content_c

The updated response output looks like the following:

```
Hello World v1.1
```

Task 4. Automating canary testing

When code is released to production, it's common to release to a small subset of live traffic before migrating all traffic to the new code base.

In this section, you implement a trigger that is activated when code is committed to the main branch. The trigger deploys the code to a unique canary URL and routes 10% of all live traffic to it.

1. In Cloud Shell, set up the branch trigger:

```
gcloud beta builds triggers create cloud-source-repositories --
trigger-config master-trigger.json
```

content_c

2. To review the new trigger, go to the **Cloud Build > Triggers** page.

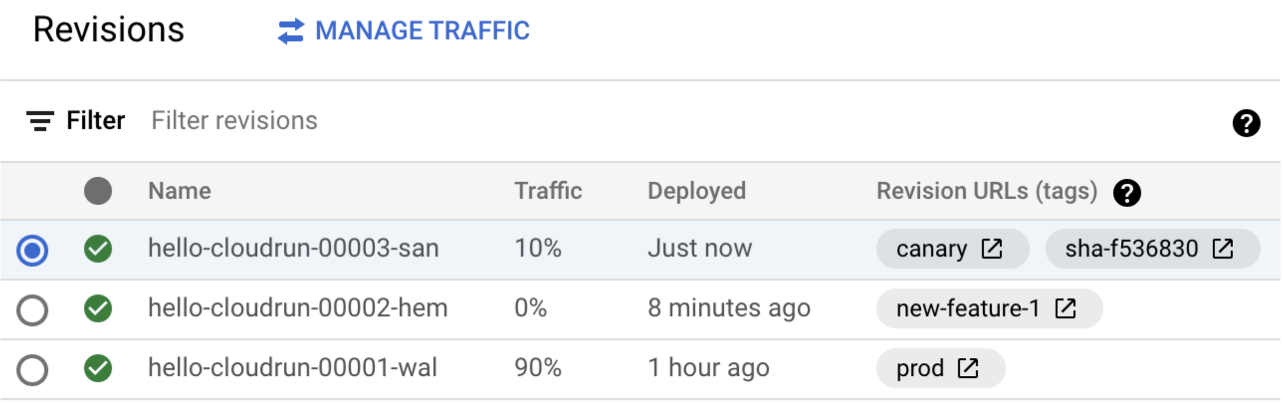
3. In Cloud Shell, merge the branch to the main line and push to the remote repository:

```
git checkout master
git merge new-feature-1
git push gcp master
```

content_c

4. To review the build in progress, go back to the **Cloud Build** page and view the current build.

5. After the build completes, to review the new revision, go to **Cloud Run**, choose the **hello-cloudrun service** , and select the **Revisions** page. Note that 90% of the traffic is routed to prod, 10% to canary, and 0% to the branch revisions.



	●	Name	Traffic	Deployed	Revision URLs (tags)
◉	✓	hello-cloudrun-00003-san	10%	Just now	canary sha-f536830
○	✓	hello-cloudrun-00002-hem	0%	8 minutes ago	new-feature-1
○	✓	hello-cloudrun-00001-wal	90%	1 hour ago	prod

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

○ Create the master trigger and build a new revision

Check my progress

6. Review the key lines of `master-cloudbuild.yaml` that implement the logic for the canary deploy.

Lines 39-44 deploy the new revision and use the tag flag to route traffic from the unique canary URL:

```
gcloud run deploy ${_SERVICE_NAME} \
--platform managed \
--region ${_REGION} \
--image gcr.io/${PROJECT_ID}/${_SERVICE_NAME} \
--tag=canary \
--no-traffic
```

Line 61 adds a static tag to the revision that notes the Git short SHA of the deployment:

```
gcloud beta run services update-traffic ${_SERVICE_NAME} --update-tags=sha-$SHORT_SHA=$${CANARY} --platform managed --region ${_REGION}
```

Line 62 updates the traffic to route 90% to production and 10% to canary:

```
gcloud run services update-traffic ${_SERVICE_NAME} --to-revisions=$${PROD}=90,$${CANARY}=10 --platform managed --region ${_REGION}
```

7. In Cloud Shell, get the unique URL for the canary revision:

```
CANARY_URL=$(gcloud run services describe hello-cloudrun --platform managed --region $REGION --format=json | jq --raw-output ".status.traffic[] | select (.tag==\"canary\")|.url")
echo $CANARY_URL
```

content_c

8. Review the canary endpoint directly:

```
curl -H "Authorization: Bearer $(gcloud auth print-identity-token)" $CANARY_URL
```

content_c

9. To see percentage-based responses, make a series of requests:

```
LIVE_URL=$(gcloud run services describe hello-cloudrun --
platform managed --region $REGION --format=json | jq --raw-
output ".status.url")
for i in {0..20};do
curl -H "Authorization: Bearer $(gcloud auth print-identity-
token)" $LIVE_URL; echo \n
done
```

content_c

Task 5. Releasing to Production

After the canary deployment is validated with a small subset of traffic, you release the deployment to the remainder of the live traffic.

In this section, you set up a trigger that is activated when you create a tag in the repository. The trigger migrates 100% of traffic to the already deployed revision based on the commit SHA of the tag. Using the commit SHA ensures the revision validated with canary traffic is the revision utilized for the remainder of production traffic.

1. In Cloud Shell, set up the tag trigger:

```
gcloud beta builds triggers create cloud-source-repositories --
trigger-config tag-trigger.json
```

content_c

2. To review the new trigger, go to the Cloud Build Triggers page (https://console.cloud.google.com/cloud-build/triggers) in the Cloud console.

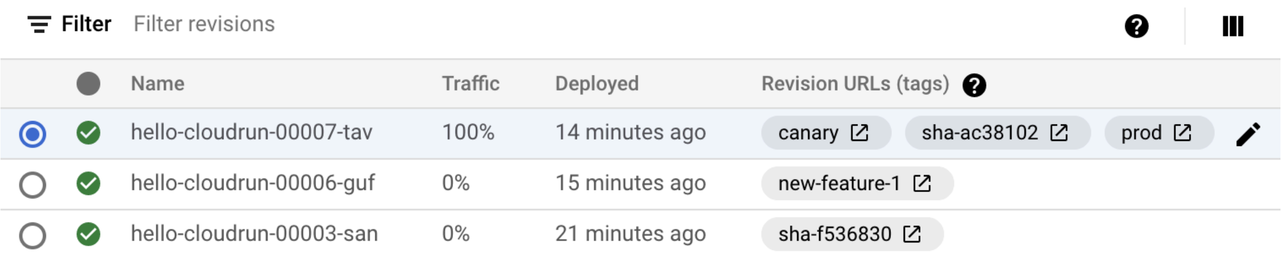
3. In Cloud Shell, create a new tag and push to the remote repository:

```
git tag 1.1
git push gcp 1.1
```

content_c

4. To review the build in progress, go to the Cloud Build Builds page (https://console.cloud.google.com/cloud-build/builds) in the Cloud console.

5. After the build is complete, to review the new revision, go to the **Cloud Run**, choose the **hello-cloudrun service**, and select the **Revisions** page in the Cloud console. Note that the revision is updated to indicate the prod tag and it is serving 100% of live traffic.



	●	Name	Traffic	Deployed	Revision URLs (tags)
◉	✓	hello-cloudrun-00007-tav	100%	14 minutes ago	canary sha-ac38102 prod
○	✓	hello-cloudrun-00006-guf	0%	15 minutes ago	new-feature-1
○	✓	hello-cloudrun-00003-san	0%	21 minutes ago	sha-f536830

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

○ Create the tag trigger and view the updated revision

Check my progress

6. In Cloud Shell, to see percentage-based responses, make a series of requests:

```
LIVE_URL=$(gcloud run services describe hello-cloudrun --
platform managed --region $REGION --format=json | jq --raw-
output ".status.url")
for i in {0..20};do
curl -H "Authorization: Bearer $(gcloud auth print-identity-
token)" $LIVE_URL; echo \n
done
```

content_c

7. Review the key lines of `tag-cloudbuild.yaml` that implement the production deployment logic.

Line 37 updates the canary revision adding the prod tag. The deployed revision is now tagged for both prod and canary:

```
gcloud beta run services update-traffic ${_SERVICE_NAME} --update-tags=prod=$${CANARY} --platform managed --region ${_REGION}
```

Line 39 updates the traffic for the base service URL to route 100% of traffic to the revision tagged as prod:

```
gcloud run services update-traffic ${_SERVICE_NAME} --to-revisions=$${NEW_PROD}=100 --platform managed --region ${_REGION}
```

Congratulations!

Congratulations! In this lab, you learned how to implement a deployment pipeline for Cloud Run that executes a progression of code from developer branches to production with automated canary testing and percentage based traffic management. Now you can use Cloud Build to create and rollback continuous integration pipelines with Cloud Run on Google Cloud!

Next steps / Learn more

- Review documentation for Managing Revisions with CloudRun (https://cloud.google.com/run/docs/managing/revisions).
- Read the CloudRun documentation for Rollbacks, gradual rollouts, and traffic migration (https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration).
- Review the documentation for Using tags for accessing revisions (https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration#tags).
- Learn more about Creating and managing build triggers (https://cloud.google.com/build/docs/automating-builds/create-manage-triggers) in CloudBuild.

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. Our classes (https://cloud.google.com/training/courses) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications (https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated February 5, 2024

Lab Last Tested August 30, 2023