Start Lab 02:15:00

Continuous Delivery with Google Cloud Deploy

Lab 1 hour 30 minutes No cost Intermediate

★★★★★

GSP1079

Overview Google Cloud Deploy

Lab 1 hour 30 minutes No cost Intermediate

★★★★★

GSP1079

Google Cloud Self-Paced Labs

Overview Google Cloud Deploy

Lab 1 hour 30 minutes No cost Intermediate

★★★★★

GSP1079

Google Cloud Self-Paced Labs

Google Cloud Deploy

Lab 1 hour 30 minutes No cost Intermediate

GSP1079

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"
   ```

   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"
   ```

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

6. Click **Next**.

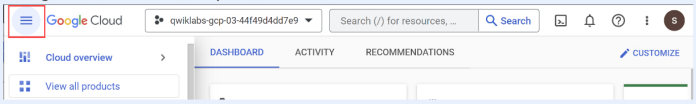
6. Click **Next**.

> **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
```

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
```

Output:

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

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Task 1. Set variables

- Declare the environment variables that will be used by various commands:

```
export PROJECT_ID=$(gcloud config get-value project)
export REGION=""
gcloud config set compute/region $REGION
```

Task 2. Create three GKE clusters

In this task you will create the three GKE clusters that will be targets for the delivery pipeline.

Three GKE clusters will be created, denoting the three targets for the delivery pipeline:

- **test**
- **staging**
- **prod**

1. Enable the Google Kubernetes Engine API:

```
gcloud services enable \
container.googleapis.com \
clouddeploy.googleapis.com
```

2. Create the three GKE clusters:

```
gcloud container clusters create test --node-locations="" --num-
nodes=1  --async
gcloud container clusters create staging --node-locations="" --
num-nodes=1  --async
gcloud container clusters create prod --node-locations="" --num-
nodes=1  --async
```

3. Check the status of the three clusters:

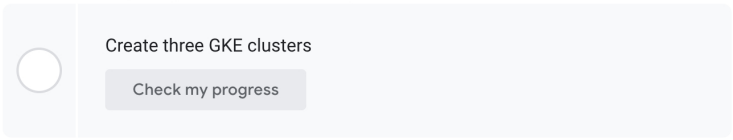
```
gcloud container clusters list --format="csv(name,status)"
```

Output

```
name,status
prod,PROVISIONING
staging,PROVISIONING
test,RUNNING
```

Creating the clusters can take a few minutes. You don't need to wait for the clusters to be ready. Continue the lab.

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



Task 3. Prepare the web application container image

In this task you'll create a repository in Artifact Registry to hold the web application's container images.

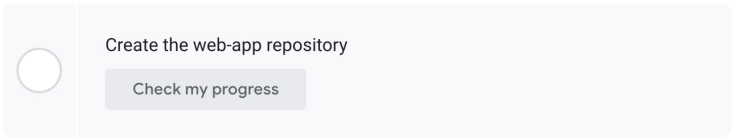
1. Enable the Artifact Registry API:

```
gcloud services enable artifactregistry.googleapis.com
```

2. Create the web-app repository for holding container images:

```
gcloud artifacts repositories create web-app \
--description="Image registry for tutorial web app" \
--repository-format=docker \
--location=$REGION
```

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



Task 4. Build and deploy the container images to the Artifact Registry

In this task you will clone the git repository containing the web application and deploy the application's container images to Artifact Registry.

Prepare the application configuration

1. Clone the repository for the lab into your home directory:

```
cd ~/
git clone https://github.com/GoogleCloudPlatform/cloud-deploy-tutorials.git
cd cloud-deploy-tutorials
```

```
cd cloud-deploy-tutorials
git checkout c3cae80 --quiet
cd tutorials/base
```

2. Create the `skaffold.yaml` configuration:

```
envsubst < clouddeploy-config/skaffold.yaml.template >
web/skaffold.yaml
cat web/skaffold.yaml
```

The web directory now contains the `skaffold.yaml` configuration file, which provides instructions for Skaffold to build a container image for your application. This configuration describes the following items.

The build section configures:

- The two container images that will be built (artifacts)
- The Google Cloud Build project used to build the images

The `deploy` section configures the Kubernetes manifests needed in deploying the workload to a cluster.

The `portForward` configuration is used to define the Kubernetes service for the deployment.

Output

```
apiVersion: skaffold/v2beta7
kind: Config
build:
  artifacts:
    - image: leeroy-web
      context: leeroy-web
    - image: leeroy-app
      context: leeroy-app
  googleCloudBuild:
    projectId: {{project-id}}
deploy:
  kubectl:
    manifests:
      - leeroy-web/kubernetes/*
      - leeroy-app/kubernetes/*
portForward:
  - resourceType: deployment
    resourceName: leeroy-web
    port: 8080
    localPort: 9000
```

Note: To view the files, use vi, emacs, nano or the Cloud Shell Code Editor by clicking on the **Open Editor** icon in Cloud Shell.

Build the web application

The skaffold tool will handle submission of the codebase to Cloud Build.

1. Enable the Cloud Build API:

```
gcloud services enable cloudbuild.googleapis.com
```

2. Run the skaffold command to build the application and deploy the container image to the Artifact Registry repository previously created:

```
cd web
```

```
skaffold build --interactive=false \
--default-repo $REGION-docker.pkg.dev/$PROJECT_ID/web-app \
--file-output artifacts.json
cd ..
```

3. Once the skaffold build has completed, check for the container images in Artifact Registry:

```
gcloud artifacts docker images list \
$REGION-docker.pkg.dev/$PROJECT_ID/web-app \
--include-tags \
--format yaml
```

The `--format` yaml parameter returns the output as YAML for readability. The output should look like this:

Output

```
---
createTime: '2022-01-14T02:07:54.995807Z'
package: us-central1-docker.pkg.dev/{{project-id}}/web-app/leeroy-app
tags: '9181623'
updateTime: '2022-01-14T02:07:54.995807Z'
version:
sha256:6af6a0a72d13dd6597c0fc0191f697e2da2c3892d1bf8e87a3df8d96612e1495
---
createTime: '2022-01-14T02:07:53.629263Z'
package: us-central1-docker.pkg.dev/{{project-id}}/web-app/leeroy-web
tags: '9181623'
updateTime: '2022-01-14T02:07:53.629263Z'
version:
sha256:a0179673d1876f205875b223557c83162e56e91c5e3313f5e99465a224adb6c9
```

By default, Skaffold sets the tag for an image to its related git tag if one is available. Similar information can be found in the `artifacts.json` file that was created by the skaffold command.

Skaffold generates the `web/artifacts.json` file with details of the deployed images:

```
cat web/artifacts.json | jq
```

Output

```
{
    "builds": [
        {
            "imageName": "leeroy-web",
            "tag": "us-central1-docker.pkg.dev/{{project-id}}/web-
app/leeroy-
web:9181623@sha256:a0179673d1876f205875b223557c83162e56e91c5e3313f5e99465a
        },
        {
            "imageName": "leeroy-app",
            "tag": "us-central1-docker.pkg.dev/{{project-id}}/web-
app/leeroy-
app:9181623@sha256:6af6a0a72d13dd6597c0fc0191f697e2da2c3892d1bf8e87a3df8d9
        }
    ]
```

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

Build and deploy the container images to the Artifact Registry

Check my progress

Task 5. Create the delivery pipeline

In this task you will set up the delivery pipeline.

1. Enable the Google Cloud Deploy API:

```
gcloud services enable clouddeploy.googleapis.com
```

2. Create the delivery-pipeline resource using the `delivery-pipeline.yaml` file:

```
gcloud config set deploy/region $REGION
cp clouddeploy-config/delivery-pipeline.yaml.template
clouddeploy-config/delivery-pipeline.yaml
gcloud beta deploy apply --file=clouddeploy-config/delivery-
pipeline.yaml
```

3. Verify the delivery pipeline was created:

```
gcloud beta deploy delivery-pipelines describe web-app
```

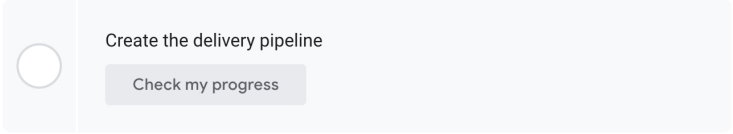
The delivery pipeline will appear similar to the following output:

Output

```
Unable to get target test
Unable to get target staging
Unable to get target prod
Delivery Pipeline:
  createTime: '2021-08-16T14:03:18.294884547Z'
  description: web-app delivery pipeline
  etag: 2539eacd7f5c256d
  name: projects/{{project-id}}/locations/us-
central1/deliveryPipelines/web-app
  serialPipeline:
    stages:
    - targetId: test
    - targetId: staging
    - targetId: prod
  uid: eb0601aa03ac4b088d74c6a5f13f36ae
  updateTime: '2021-08-16T14:03:18.680753520Z'
Targets: []
```

Notice the first three lines of the output. The delivery pipeline currently references three target environments that haven't been created yet. In the next task you will create those targets.

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



Task 6. Configure the deployment targets

Three delivery pipeline targets will be created - one for each of the GKE clusters.

Ensure that the clusters are ready

The three GKE clusters should now be running, but it's useful to verify this.

- Run the following to get the status of the clusters:

```
gcloud container clusters list --format="csv(name,status)"
```

All three clusters should be in the RUNNING state, as indicated in the output below. If they are not yet marked as RUNNING, retry the command above until their status has changed to RUNNING.

Output

```
name,status
prod,RUNNING
staging,RUNNING
test,RUNNING
```

Once all the clusters have the "RUNNING" status continue the lab.

Create a context for each cluster

Use the commands below to get the credentials for each cluster and create an easy-to-use `kubectl` context for referencing the clusters later:

```
CONTEXTS=("test" "staging" "prod")
for CONTEXT in ${CONTEXTS[@]}
do
    gcloud container clusters get-credentials ${CONTEXT} --
region ${REGION}
    kubectl config rename-context
gke_${PROJECT_ID}_${REGION}_${CONTEXT} ${CONTEXT}
done
```

Create a namespace in each cluster

Use the commands below to create a Kubernetes namespace (web-app) in each of the three clusters:

```
for CONTEXT in ${CONTEXTS[@]}
do
    kubectl --context ${CONTEXT} apply -f kubernetes-config/web-
app-namespace.yaml
done
```

The application will be deployed to the (web-app) namespace.

Create the delivery pipeline targets

1. Submit a target definition for each of the targets:

```
for CONTEXT in ${CONTEXTS[@]}
do
    envsubst < clouddeploy-config/target-$CONTEXT.yaml.template
```

```
> clouddeploy-config/target-$CONTEXT.yaml
    gcloud beta deploy apply --file clouddeploy-
config/target-$CONTEXT.yaml
done
```

The targets are described in a yaml file. Each target configures the relevant cluster information for the target. The test and staging target configurations are mostly the same.

2. Display the details for the test Target:

```
cat clouddeploy-config/target-test.yaml
```

Output

```
apiVersion: deploy.cloud.google.com/v1beta1
kind: Target
metadata:
  name: test
description: test cluster
gke:
  cluster: projects/{{project-id}}/locations/us-central1/clusters/test
```

The prod target is slightly different as it requires approval (see the `requireApproval` setting in the output) before a release can be promoted to the cluster.

3. Display the details for the prod Target:

```
cat clouddeploy-config/target-prod.yaml
```

Output

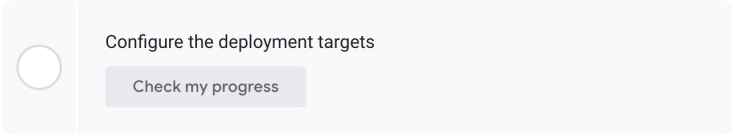
```
apiVersion: deploy.cloud.google.com/v1beta1
kind: Target
metadata:
  name: prod
description: prod cluster
requireApproval: true
gke:
  cluster: projects/{{project-id}}/locations/us-central1/clusters/prod
```

4. Verify the three targets (test, staging, prod) have been created:

```
gcloud beta deploy targets list
```

All Google Cloud Deploy targets for the delivery pipeline have now been created.

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



Task 7. Create a release

In this task you create a release of the application.

A Google Cloud Deploy release is a specific version of one or more container images associated with a specific delivery pipeline. Once a release is created, it can be promoted through multiple targets (the promotion sequence). Additionally, creating a release renders your application using skaffold and saves the output as a point-in-time reference that's used for the duration of that release.

Since this is the first release of your application, you'll name it `web-app-001`.

1. Run the following command to create the release:

```
gcloud beta deploy releases create web-app-001 \
--delivery-pipeline web-app \
--build-artifacts web/artifacts.json \
--source web/
```

The `--build-artifacts` parameter references the `artifacts.json` file created by skaffold earlier. The `--source parameter` references the application source directory where skaffold.yaml can be found.

When a release is created, it will also be automatically rolled out to the first target in the pipeline (unless approval is required, which will be covered in a later step of this lab).

2. To confirm the test target has your application deployed, run the following command:

```
gcloud beta deploy rollouts list \
--delivery-pipeline web-app \
--release web-app-001
```

Output

```
---
approvalState: DOES_NOT_NEED_APPROVAL
createTime: '2021-08-16T14:05:21.961604Z'
deployEndTime: '2021-08-16T14:06:35.278604Z'
deployStartTime: '2021-08-16T14:06:22.420091744Z'
deployingBuild: projects/{{project-id}}/locations/us-
central1/builds/4815b788-ec5e-4185-9141-a5b57c71b001
enqueueTime: '2021-08-16T14:06:21.760830Z'
etag: 5cb7b6c342b5f29b
name: projects/{{project-id}}/locations/us-
central1/deliveryPipelines/web-app/releases/web-app-001/rollouts/web-
app-001-to-test-0001
state: SUCCESS
targetId: test
uid: cccd9525d3a0414fa60b2771036841d9
```

The first rollout of a release will take several minutes because Google Cloud Deploy renders the manifests for all targets when the release is created. The GKE cluster may also take a few minutes to provide the resources required by the deployment.

If you do not see `state: SUCCESS` in the output from the previous command, please wait and periodically re-run the command until the rollout completes.

3. Confirm your application was deployed to the test GKE cluster by running the following commands:

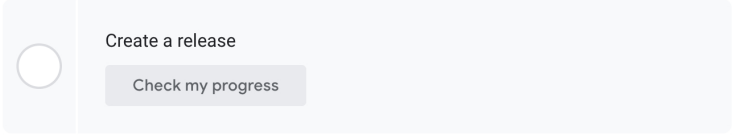
```
kubectx test
kubectl get all -n web-app
```

Output

```
NAME                              READY   STATUS    RESTARTS   AGE
pod/leeroy-app-5547cf9d9b-rgc2l   1/1     Running   0          3m27s
pod/leeroy-web-6768b49c46-w7vt9   1/1     Running   0          3m27s
```

```
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
service/leeroy-app   ClusterIP   None         <none>        50051/TCP
3m28s
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/leeroy-app   1/1     1            1           3m28s
deployment.apps/leeroy-web   1/1     1            1           3m28s
NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/leeroy-app-5547cf9d9b   1         1         1
3m28s
replicaset.apps/leeroy-web-6768b49c46   1         1         1
3m28s
```

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



Task 8. Promote the application to staging

In this task you will promote the application from test and into the staging target.

1. Promote the application to the staging target:

```
gcloud beta deploy releases promote \
--delivery-pipeline web-app \
--release web-app-001
```

You will be prompted to continue before the promotion commences.

- Press ENTER to accept the default (Y = yes).

2. To confirm the staging Target has your application deployed, run the following command:

```
gcloud beta deploy rollouts list \
--delivery-pipeline web-app \
--release web-app-001
```

Review the output

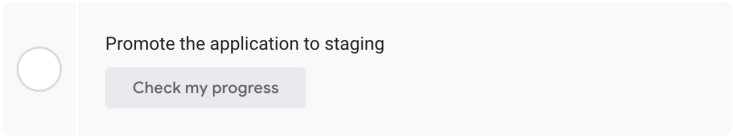
Look for the section marked `targetId: staging`. As before, if you do not see `state: SUCCEEDED` in the output from the previous command, wait and periodically re-run the command until the rollout completes.

Output

```
---
approvalState: DOES_NOT_NEED_APPROVAL
createTime: '2022-01-05T02:19:32.539468Z'
deployEndTime: '2022-01-05T02:19:45.970949Z'
deployStartTime: '2022-01-05T02:19:33.111948770Z'
deployingBuild: projects/743805075658/locations/us-
central1/builds/2316517c-3a2f-4cd3-80ad-6d133b653746
etag: 1109b802ff586df5
name: projects/{{project-id}}/locations/us-
central1/deliveryPipelines/web-app/releases/web-app-001/rollouts/web-
app-001-to-staging-0001
```

```
app-001-to-staging-0001
state: SUCCEEDED
targetId: staging
uid: 80a35a5f044844708d2050f8c556e07e
```

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



Task 9. Promote the application to prod

In this task you will again promote the application but will also provide approval.

1. Promote the application to the prod target:

```
gcloud beta deploy releases promote \
--delivery-pipeline web-app \
--release web-app-001
```

You will be prompted to continue before the promotion commences.

- Press ENTER to accept the default (Y = yes).

2. To review the status of the prod target, run the following command:

```
gcloud beta deploy rollouts list \
--delivery-pipeline web-app \
--release web-app-001
```

In the output, note that the `approvalState` is `NEEDS_APPROVAL` and the state is `PENDING_APPROVAL`.

Output

```
---
approvalState: NEEDS_APPROVAL
createTime: '2021-08-16T14:12:07.466989Z'
etag: 6e9303e5a1b04084
name: projects/{{project-id}}/locations/us-
central1/deliveryPipelines/web-app/releases/web-app-001/rollouts/web-
app-001-to-prod-0001
state: PENDING_APPROVAL
targetId: prod
uid: a5c7d6007fee4d80904d49142581aaa7
```

3. Approve the rollout with the following:

```
gcloud beta deploy rollouts approve web-app-001-to-prod-0001 \
--delivery-pipeline web-app \
--release web-app-001
```

You will be prompted to approve the rollout before the promotion commences.

- Press ENTER to accept the default (Y = yes).

4. To confirm the prod target has your application deployed, run the following command:

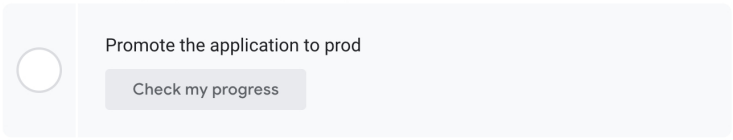
```
gcloud beta deploy rollouts list \
--delivery-pipeline web-app \
--release web-app-001
```

As for previous rollouts, locate the entry for the target (`targetId: prod`) and check that the rollout has completed (`state: SUCCEEDED`). Periodically re-run the command until the rollout completes.

5. Use `kubectl` to check on the status of the deployed application:

```
kubectx prod
kubectl get all -n web-app
```

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



Congratulations!

Congratulations! In this lab, you learned how to create a delivery pipeline using Google Cloud Deploy. You created a release for a basic application and promoted the application through a series of Google Kubernetes Engine (GKE) targets. You first deployed the application to the test target, then promoted it to the staging target, and finally to the prod target. Now you can use Cloud Deploy to create continuous delivery pipelines!

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. Our classes 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 help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated February 5, 2024

Lab Last Tested July 12, 2023