Implement DevOps in Google Cloud: Challenge Lab

GSP330

Overview

In a challenge lab you're given a scenario and a set of tasks. Instead of following step-by-step instructions, you will use the skills learned from the labs in the quest to figure out how to complete the tasks on your own! An automated scoring system (shown on this page) will provide feedback on whether you have completed your tasks correctly.

When you take a challenge lab, you will not be taught new Google Cloud concepts. You are expected to extend your learned skills, like changing default values and reading and researching error messages to fix your own mistakes.

To score 100% you must successfully complete all tasks within the time period!

This lab is recommended for students enrolled in the Implement DevOps in the Google Cloud Quest. Are you ready for the challenge?
Topics tested

- Use Jenkins console logs to resolve application deployment issues.
- Deploy and a development update to a sample application for Jenkins to deply using a development pipeline.
- Deploy and test a Kubernetes Canary deployment for a sample application.
- Push the Canary application branch to master and confirm this triggers a production pipeline update.

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 Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access 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 Pixelbook, open an Incognito window to run this lab.

You must wait for the lab setup to finish before making any changes. The Kubernetes cluster and Jenkins environment are setup automatically for you when the lab is started and that process takes about five minutes. While you wait for the setup to complete you should use the time to read through the scenario, explore the lab environment, and take note of the tips and tricks at the end of the instructions.

Challenge scenario

You have started a new role as a Junior Cloud Engineer for Jooli Inc. You're expected to help manage the Cloud infrastructure and deployment tools at Jooli. Common tasks include provisioning resources for projects and implementing new products and services to help improve Jooli's service management in real time.

You are expected to have the skills and knowledge for these tasks, so step-by-step guides won't be provided.

Some Jooli Inc. standards you should follow:

- Create all resources in the `us-east1` region and `us-east1-b` zone, unless otherwise directed.
- Use the project VPCs.
- Naming is normally *team-resource*, e.g. an instance could be named **kraken-webserver1**.
- Allocate cost effective resource sizes. Projects are monitored and excessive resource use will result in the containing project's termination (and possibly yours), so beware. This is the guidance the monitoring team is willing to share; unless directed, use `n1-standard-1`.

Your challenge

As soon as you sit down at your desk and open your new laptop you receive the following request to complete a set of tasks. Good luck!

You must wait for the lab to provision before making any changes to the environment! The pre-configured parts of the environment that you need to work with will be available to you as soon as the lab indicates it is ready.

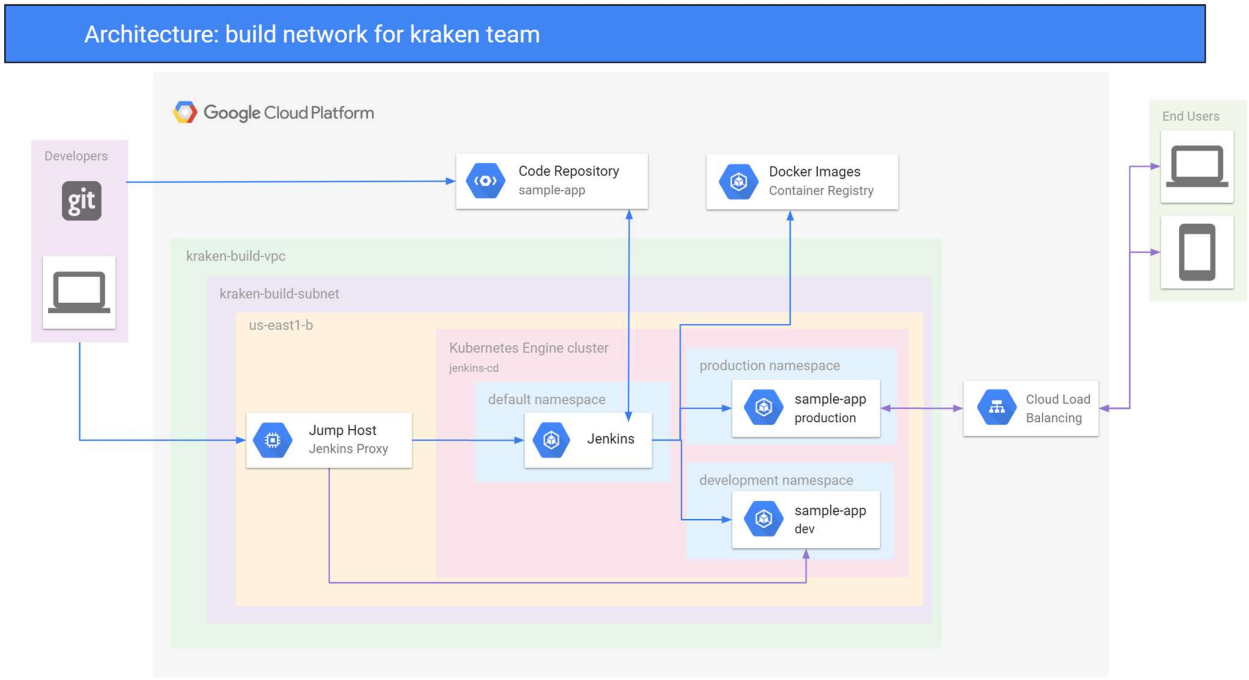
Task 1: Configure a Jenkins pipeline for continuous deployment to Kubernetes Engine

Your first task is to create a multibranch pipeline using Jenkins that automatically deploys new versions of the test application, **sample-app**.

A VPC build environment, called **kraken-vpc**, has been created for you to create and test continuous deployment pipelines using the Jenkins build and deployment automation tool. Inside that VPC, you must use the newly deployed Kubernetes cluster, called **jenkins-cd**. Jenkins is already pre-installed and running in that cluster (**jenkins-cd**) but no other configuration has taken place.

As this is a test environment, the architecture is compact. The diagram below describes the VPC network and Kubernetes Engine environment that has been created for you. You will use Jenkins to deploy different versions of the application to various Kubernetes

namespaces inside the **jenkins-cd** cluster, using different sets of Kubernetes configuration manifest files for production, canary, and deployment build and deployment procedures.



The application, **sample-app**, has been cloned into a **Cloud Source Repository** as part of your project and a Jenkins job configuration file, called **Jenkinsfile**, is included in the root of that repository. You use **sample-app** to test the Jenkins CI/CD pipeline behavior and then use **Jenkinsfile** to manage automated build and deployment pipelines for the application. You can explore the component files directly from the repository. You can examine the Kubernetes manifest templates for the three types of deployment (production, canary, and dev) in the `sample-app/k8s` directory.

A Jumphost, called `kraken-jumphost`, has been automatically created and then used to create the initial version of **sample-app**. The initial version of **sample-app** has been automatically pushed to your Cloud Source Repository. The application's source files are in a directory called `/work/sample-app`. With this jumphost you access the Jenkins admin interface to configure and test your continuous deployment pipeline.

To configure a Jenkins pipeline and deploy the initial version of the application, you must complete the following initial tasks. Additional tasks may be required for successful deployment.

- Configure `kubectl proxy` to access the Jenkins UI.
- Retrieve and decode the Jenkins Admin password (key name `jenkins-admin-password`) for your deployment from the Kubernetes secret (secret name `cd-jenkins`) for the Jenkins deployment.
- Configure Jenkins to get the build credentials required from the metadata available on the Kubernetes Engine nodes. The cluster has been deployed using a service account that has the required read and write access to Cloud Source Repositories.
- Configure a multibranch Jenkins pipeline job using the **Jenkinsfile** Jenkins job configuration file included in the root of the **sample-app** repository.

- Configure the Jenkins job to check for new branches / repository updates every minute. Once you successfully configure the Jenkins pipeline, it should immediately process the contents of the repository that it's configured to read from and deploy the initial version of the application contained in your repository.

  If the pipeline fails to deploy successfully, examine the console output from the Jenkins pipeline to identify issues that need to be resolved.

  Click *Check my progress* to verify the objective.

  If you don't get a green check mark, click on the **Score** fly-out on the top right and click **Run Step** on the relevant step. A hint pop up opens to give you advice.
  Once the Jenkins pipeline has completed, test the application using the external ip-address of the production frontend service. The initial version of the application displays a blue border and reports a version number of v1.0.0.

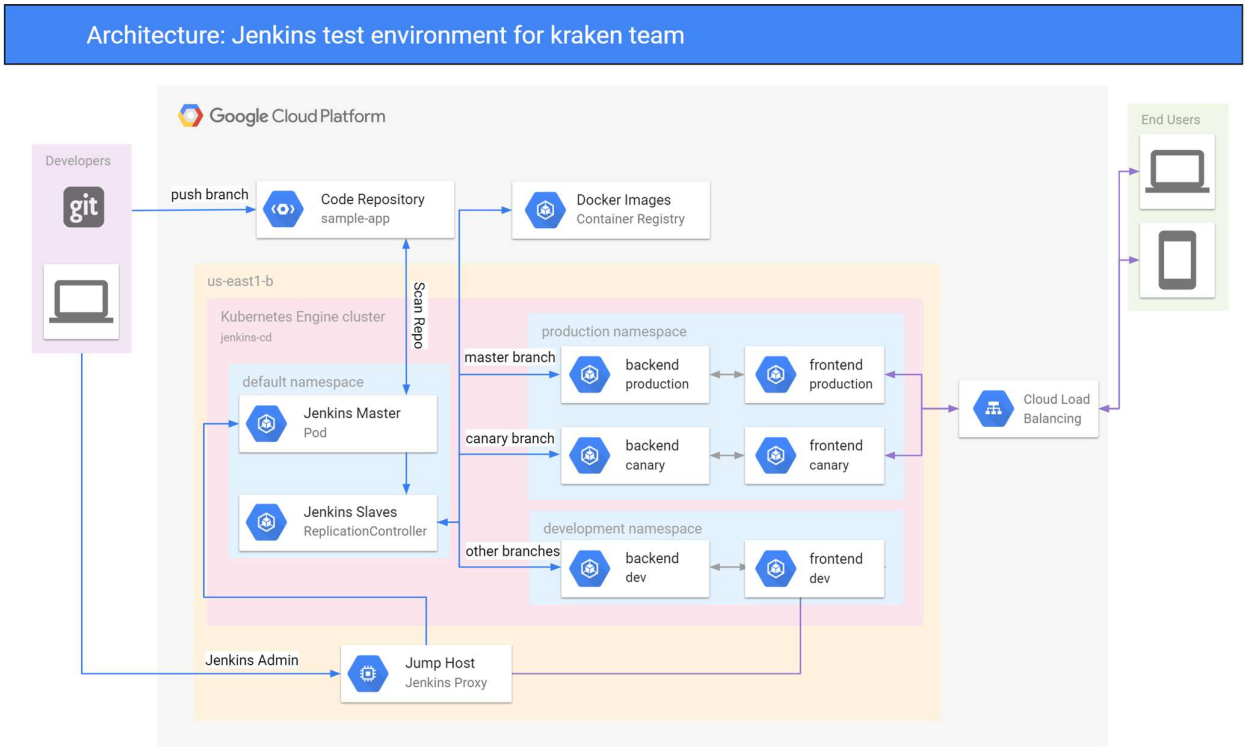
Task 2: Push an update to the application to a development branch

Your second task is to makes some changes to the application and use those to make sure that Jenkins automatically detects and deploys new development builds correctly.

The Jenkins pipeline configuration contained in the `Jenkinsfile` file in the root directory of the `sample-app` repository is configured to behave as follows:

- Changes to the **master** branch trigger a Jenkins job that deploys a production update in the production namespace using the `production/*.yaml` templates.
- Changes to the **canary** branch trigger a Jenkins job that deploys a canary update in the production namespace using the `canary/*.yaml` templates.
- Changes to any other branch trigger a Jenkins job that deploys the application to a namespace that matches the branch name used using the `dev/*.yaml` templates.

Both the canary and production deployments are exposed to external traffic using a Kubernetes load balancer service. Development deployments are not exposed externally and must be tested from a system with access into the cluster, such as the kraken jumphost. For this task you must create a new branch for development and push the changes back to the repository.



Make sure you have **sample-app** cloned from the Cloud Source repository for your project. The `/work/sample-app` directory in the `kraken-jumphost` compute instance has been cloned from that repository using git.

Update the application to report v2.0.0 and display an orange background.

- Edit the file `main.go` in the root folder of the repo and change the version number to "2.0.0".

```
const version string = "2.0.0"
```

- Edit the file `html.go` in the root folder of the repo and change both lines that contains the word `blue` to `orange`.

```
<div class="card orange">
```

You must now create a new development branch with a corresponding Kubernetes namespace. Once the files have been changed and the namespace exists, you can commit your changes to the development branch and push those changes to the repository.

Jenkins should detect the change and deploy a development version of the application.

Click *Check my progress* to verify the objective.

It doesn't look like you've completed this step yet. Try again.

If you don't get a green check mark, click on the **Score** fly-out on the top right and click **Run Step** on the relevant step. A hint pop up opens to give you advice.

You cannot access development builds from outside the cluster because the Kubernetes service configuration file used by the Jenkins configuration file for development builds does not create an externally accessible frontend service.

Task 3: Push a Canary deployment to the production namespace

Now that you're satisfied that the development deployment was successfully deployed by Jenkins, you must deploy and test it as a Canary deployment. As you can see from the application diagram in the previous task, canary deployments can be accessed using the external load balancer address for the application.

Now create a new branch called canary, merge the development branch with it, and push that to the repository.

If you have configured the Jenkins multibranch pipeline correctly, a canary build will be deployed that can be tested using the production load balancer.

If the canary deployment is successful, the external load balancer address switches to servicing the new version, reporting v2.0.0 with an orange theme, for some requests.

Click *Check my progress* to verify the objective.

If you don't get a green check mark, click on the **Score** fly-out on the top right and click **Run Step** on the relevant step. A hint pop up opens to give you advice.

Task 4: Promote the Canary Deployment to production

Now that the Canary deployment has been successfully tested, promote the Canary to production.

Merge the Canary branch with master in order to trigger a Jenkins build for the updated master branch. If this is successful then all requests that are made to the production load balancer should now report the updated version 2.0.0 and use the orange theme.

Click *Check my progress* to verify the objective.

If you don't get a green check mark, click on the **Score** fly-out on the top right and click **Run Step** on the relevant step. A hint pop up opens to give you advice.

Tips and Tricks

- **Tip 1** The VPC networks and Kubernetes cluster have been created for you but the Kubernetes namespaces required by your Jenkins configuration have not been created for you. You must create the production namespace that the Jenkins configuration file requires before pushing changes that will trigger a Jenkins production or canary build.

- **Tip 2** The name of the Jenkins secret that contains the admin username and password keys (`cd-jenkins`) is not the same as the name of the cluster (**jenkins-cd**).

- **Tip 3** The git repository URL for your sample application is in the form `https://source.developers.google.com/p/[PROJECT_ID]/r/sample-app` where `[PROJECT_ID]` must be replaced by your lab project ID.

- **Tip 4** The Jenkins UI is not connected to the internet directly. You must configure a Kubernetes proxy using kubectl to connect to the Jenkins Master Pod on port 8080.

- **Tip 5** The Development deployments are not exposed to external traffic for security reasons. In order to test the Development version of the application you will need to connect to the Kubernetes cluster using `kubectl proxy` and then query the front-end service via `localhost`.

Congratulations!

Earn Your Next Skill Badge

This self-paced lab is part of the Implement DevOps in Google Cloud skill badge quest. Completing this skill badge quest earns you the badge above, to recognize your achievement. Share your badge on your resume and social platforms, and announce your accomplishment using #GoogleCloudBadge.

Google Cloud Training & 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 23, 2021

Lab Last Tested July 29, 2020

Solutions: Video Code