**Delivery with Jenkins in Kubernetes on GCP**

**MINI PROJECT REPORT**

**18CSE316J – Essentials in Cloud and Devops**

**LABORATORY**

**(2018 Regulation)**

**III Year/ VI Semester**

**Academic Year: 2022 -2023**

By

Shivam Singh

**Abstract**

Over the past decade there has been tremendous change and advancement in how applications are built and deployed by developers as a result of the ever-increasing demands from end users. " Cloud with Kubernetes" is a book that provides an in-depth overview of Kubernetes, an open-source container and set up a continuous delivery pipeline with Jenkins on Kubernetes engine. Jenkins is the go-to automation server used by developers who frequently integrate their code in a shared repository

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Topic** | **Page No** |
| **1** | **Introduction** | **5** |
| **2** | **Literature Survey** |  |
| **3** | **Methodology** |  |
| **4** | **System Architecture** |  |
| **5** | **Deployment of other services** |  |
| **6** | **Output** |  |
| **7** | **Conclusion** |  |
| **8** | **References** |  |

**ABBREVIATIONS**

AWS : Amazon Web Services

IaC : Infrastructure as Code

Node.js : Node, a JavaScript runtime

npm : Node Package Manager, a package manager for Node.js

CLI : Command Line Interface

EC2 Elastic Compute Cloud

IAM Identity and Access Management

SG : Security Group

**INTRODUCTION**

Kubernetes Engine is Google Cloud's hosted version of Kubernetes - a powerful cluster manager and orchestration system for containers. Kubernetes is an open source project that can run on many different environments—from laptops to high-availability multi-node clusters; from virtual machines to bare metal. As mentioned before, Kubernetes apps are built on containers - these are lightweight applications bundled with all the necessary dependencies and libraries to run them. This underlying structure makes Kubernetes applications highly available, secure, and quick to deploy—an ideal framework for cloud developers.

[Jenkins](https://jenkins.io/) is an open-source automation server that lets you flexibly orchestrate your build, test, and deployment pipelines. Jenkins allows developers to iterate quickly on projects without worrying about overhead issues that can stem from continuous delivery.

**Context of project :**

As businesses increasingly move towards cloud-based solutions and microservices architecture, the need for efficient deployment and management of containerized applications becomes essential. Kubernetes, an open-source container orchestration platform, has emerged as the go-to solution for simplifying the deployment, scaling, and management of containerized applications in a cloud environment. When you need to set up a continuous delivery (CD) pipeline, deploying Jenkins on Kubernetes Engine provides important benefits over a standard VM-based deployment.

When your build process uses containers, one virtual host can run jobs on multiple operating systems. Kubernetes Engine provides ephemeral build executors—these are only utilized when builds are actively running, which leaves resources for other cluster tasks such as batch processing jobs. Another benefit of ephemeral build executors is speed—they launch in a matter of seconds.

Kubernetes Engine also comes pre-equipped with Google's global load balancer, which you can use to automate web traffic routing to your instance(s). The load balancer handles SSL termination and utilizes a global IP address that's configured with Google's backbone network—coupled with your web front, this load balancer will always set your users on the fastest possible path to an application instance.

Now that you've learned a little bit about Kubernetes, Jenkins, and how the two interact in a CD pipeline, it's time to go build one.

**Objective :**

In this lab, you will use Helm to install Jenkins from the Charts repository. Helm is a package manager that makes it easy to configure and deploy Kubernetes applications. Once you have Jenkins installed, you'll be able to set up your CI/CD pipeline.

When installing Jenkins, a values file can be used as a template to provide values that are necessary for setup.

The book aims to help readers achieve the following objectives:

1. Understand the basics of containerization and microservices architecture, and why Kubernetes is an ideal solution for managing containerized applications in a cloud environment.

2. Gain a deep understanding of Kubernetes core concepts, such as pods, deployments, services, and volumes, and how to use them effectively on GCP

3. Learn how to deploy and manage containerized applications using Kubernetes on GCP, including scaling and rolling updates.

4. Explore advanced Kubernetes topics such as security, networking, and monitoring, and how to apply them to GCP.

5. Integrate Kubernetes with other GCP services to build a complete cloud-native application ecosystem, including Google Cloud Storage, Google Cloud Load Balancing, and Google Cloud Logging.

Overall, the objective of "Jenkins in the Cloud with Kubernetes" on GCB is to equip readers with the knowledge and skills to deploy, scale, and manage containerized applications effectively on Google Cloud Platform using Kubernetes.

**Goals:**

The goal of "Jenkins the Cloud with Kubernetes" on Google Cloud Boost (GCB) is to help readers master the skills necessary to leverage Kubernetes effectively for deploying and managing containerized applications on Google Cloud Platform (GCP).

The book aims to achieve the following goals:

1. Provide a comprehensive and practical understanding of Kubernetes core concepts, such as pods, deployments, services, and volumes, and how to use them effectively on GCP.

2. Teach readers how to deploy and manage containerized applications on GCP using Kubernetes, including scaling and rolling updates.

3. Explore advanced Kubernetes topics such as security, networking, and monitoring, and how to apply them to GCP.

4. Demonstrate how to integrate Kubernetes with other GCP services to build a complete cloud-native application ecosystem, including Google Cloud Storage, Google Cloud Load Balancing, and Google Cloud Logging.

5. Help readers become proficient in using Kubernetes to build resilient and scalable cloud-native applications on GCP.

THE MAIN TASK ARE-

* Provision a Jenkins application into a Kubernetes Engine Cluster
* Set up your Jenkins application using Helm Package Manager
* Explore the features of a Jenkins application
* Create and exercise a Jenkins pipeline

**LITERATURE SURVEY**

"Jenkins Kubernetes" is a popular book on the topic of Kubernetes and cloud-native application management. It has been well-received by readers and has received positive reviews for its comprehensive coverage of Kubernetes concepts, practical guidance, and hands-on examples.

Other books on Kubernetes and cloud-native application management that are worth mentioning include:

1. "Kubernetes: Up and Running" by Brendan Burns, Joe Beda, and Kelsey Hightower: This book is an excellent resource for those looking to get started with Kubernetes. It covers the basics of Kubernetes, including deployment, scaling, and management, and provides practical examples for deploying and managing applications on Kubernetes.

2. "Cloud Native Infrastructure" by Kris Nova and Justin Garrison: This book provides a comprehensive overview of cloud-native infrastructure, including Kubernetes and other cloud-native technologies. It covers topics such as infrastructure as code, containers, and microservices, and provides practical guidance on deploying and managing applications in a cloud-native environment.

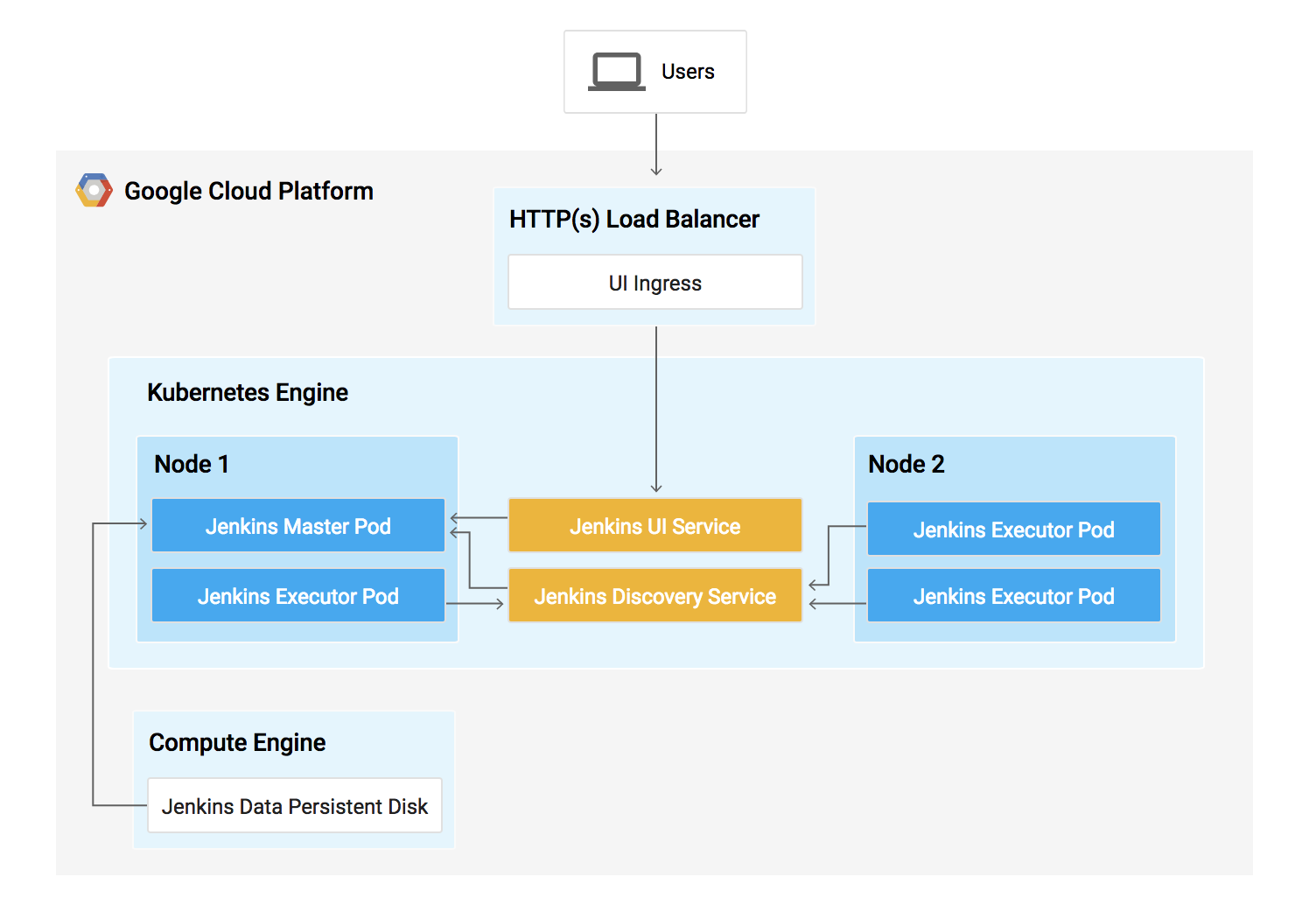
3. "Kubernetes in Action" by Marko Lukša: This book provides a deep dive into Kubernetes concepts, including pods, services, and deployments, and provides practical guidance on how to deploy and manage applications using Kubernetes. It also covers advanced topics such as security and networking.

4. "The Kubernetes Book" by Nigel Poulton: This book is a beginner-friendly guide to Kubernetes, providing an overview of its core concepts and practical examples for deploying and managing applications on Kubernetes.

**METHODOLOGY**

* Kubernetes is an open source project (available on [kubernetes.io](http://kubernetes.io/)) which can run on many different environments, from laptops to high-availability multi-node clusters, from public clouds to on-premise deployments, from virtual machines to bare metal.
* Login on GCP 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. When you are connected, you are already authenticated, and the project is set to your **PROJECT\_ID**. The output contains a line that declares the **PROJECT\_ID** for this session.
* For Google Kubernets Engine, In the cloud shell environment type the following command to set the zone and start up a cluster for use in this lab. Then ou are automatically authenticated to your cluster upon creation. If you lose connection to your Cloud Shell for any reason, run the gcloud container clusters get-credentials io command to re-authenticate. **Note:**It will take a while to create a cluster - Kubernetes Engine is provisioning a few Virtual Machines behind the scenes for you to play with! And so on ..

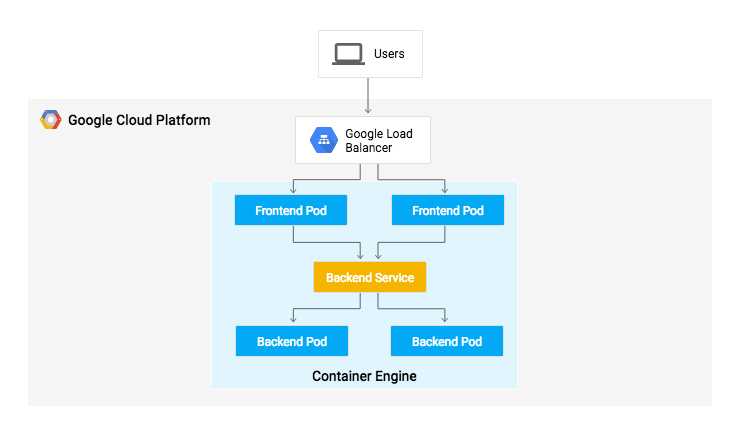
**SYSTEM ARCHITECTURE**



When you need to set up a continuous delivery (CD) pipeline, deploying Jenkins on Kubernetes Engine provides important benefits over a standard VM-based deployment.

When your build process uses containers, one virtual host can run jobs on multiple operating systems. Kubernetes Engine provides ephemeral build executors—these are only utilized when builds are actively running, which leaves resources for other cluster tasks such as batch processing jobs. Another benefit of ephemeral build executors is speed—they launch in a matter of seconds.

Kubernetes Engine also comes pre-equipped with Google's global load balancer, which you can use to automate web traffic routing to your instance(s). The load balancer handles SSL termination and utilizes a global IP address that's configured with Google's backbone network—coupled with your web front, this load balancer will always set your users on the fastest possible path to an application instance.



Pods aren't meant to be persistent. They can be stopped or started for many reasons - like failed liveness or readiness checks - and this leads to a problem:

What happens if you want to communicate with a set of Pods? When they get restarted they might have a different IP address.

That's where [Services](http://kubernetes.io/docs/user-guide/services/) come in. Services provide stable endpoints for Pods.

Services use labels to determine what Pods they operate on. If Pods have the correct labels, they are automatically picked up and exposed by our services.

The level of access a service provides to a set of pods depends on the Service's type. Currently there are three types:

* ClusterIP (internal) -- the default type means that this Service is only visible inside of the cluster,
* NodePort gives each node in the cluster an externally accessible IP and
* LoadBalancer adds a load balancer from the cloud provider which forwards traffic from the service to Nodes within it.

**DEPLOYMENT**

1. Login on gcp
2. Open cloud shell
3. For Google Kubernetes Engine enter the following command on cloud shell

gcloud config set compute/zone us-central1-b

1. For start up cluster for enter the following command

gcloud container clusters create io

1. For sample code, Copy the source code from the Cloud Shell command line:

gcloud config set compute/zone

gsutil cp gs://spls/gsp051/continuous-deployment-on-kubernetes.zip .

unzip continuous-deployment-on-kubernetes.zip

1. Now change to the correct directory

cd continuous-deployment-on-kubernetes

## Provisioning Jenkins

### **Creating a Kubernetes cluster**

gcloud container clusters create jenkins-cd \

--num-nodes 2 \

--machine-type n1-standard-2 \

--scopes <https://www.googleapis.com/auth/source.read_write,cloud-platform>

1. Before continuing, confirm that your cluster is running by executing the following command:

gcloud container clusters list

1. gcloud container clusters get-credentials jenkins-cdKubernetes Engine uses these credentials to access your newly provisioned cluster—confirm that you can connect to it by running the following command:

kubectl cluster-info

## Setup Helm

1. Add Helm's stable chart repo:

helm repo add jenkins <https://charts.jenkins.io>

1. Ensure the repo is up to date:

helm repo update

## Configure and Install Jenkins

When installing Jenkins, a values file can be used as a template to provide values that are necessary for setup.

You will use a custom values file to automatically configure your Kubernetes Cloud and add the following necessary plugins:

* Kubernetes:latest
* Workflow-multibranch:latest
* Git:latest
* Configuration-as-code:latest
* Google-oauth-plugin:latest
* Google-source-plugin:latest
* Google-storage-plugin:latest

This will allow Jenkins to connect to your cluster and your GCP project.

1. Use the Helm CLI to deploy the chart with your configuration settings:

helm install cd jenkins/jenkins -f jenkins/values.yaml –wait

1. Once that command completes ensure the Jenkins pod goes to the Running state and the container is in the READY state:

kubectl get pods

1. Configure the Jenkins service account to be able to deploy to the cluster:

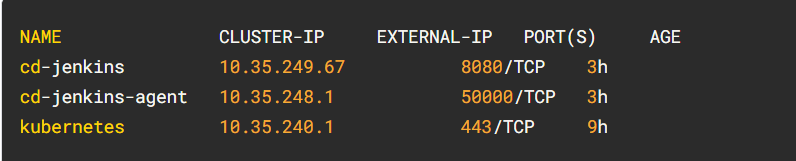
kubectl create clusterrolebinding jenkins-deploy --clusterrole=cluster-admin --serviceaccount=default:cd-jenkins

1. Run the following command to setup port forwarding to the Jenkins UI from the Cloud Shell:

export POD\_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/component=jenkins-master" -l "app.kubernetes.io/instance=cd" -o jsonpath="{.items[0].metadata.name}")

kubectl port-forward $POD\_NAME 8080:8080 >> /dev/null &

kubectl get svc



## Connect to Jenkins

1. The Jenkins chart will automatically create an admin password for you. To retrieve it, run:

printf $(kubectl get secret cd-jenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 --decode);echo

1. To get to the Jenkins user interface, click on the **Web Preview** button in Cloud Shell, then click **Preview on port 8080**:
2. If asked, log in with username admin and your auto-generated password.

## Deploying the Application

You will deploy the application into two different environments:

* **Production**: The live site that your users access.
* **Canary**: A smaller-capacity site that receives only a percentage of your user traffic. Use this environment to validate your software with live traffic before it's released to all of your users.

1. In Google Cloud Shell, navigate to the sample application directory:

cd sample-app

1. Create the Kubernetes namespace to logically isolate the deployment:

kubectl create ns production

1. Create the production and canary deployments, and the services using the kubectl apply commands:

kubectl apply -f k8s/production -n production

kubectl apply -f k8s/canary -n production

kubectl apply -f k8s/services -n production

By default, only one replica of the frontend is deployed. Use the kubectl scale command to ensure that there are at least 4 replicas running at all times.

1. Scale up the production environment frontends by running the following command:

kubectl scale deployment gceme-frontend-production -n production --replicas 4

1. Now confirm that you have 5 pods running for the frontend, 4 for production traffic and 1 for canary releases (changes to the canary release will only affect 1 out of 5 (20%) of users):

kubectl get pods -n production -l app=gceme -l role=frontend

1. Also confirm that you have 2 pods for the backend, 1 for production and 1 for canary:

kubectl get pods -n production -l app=gceme -l role=backend

1. Retrieve the external IP for the production services:

kubectl get service gceme-frontend -n production

**Note:**It can take several minutes before you see the load balancer external IP address.

1. Now, store the frontend service load balancer IP in an environment variable for use later:

export FRONTEND\_SERVICE\_IP=$(kubectl get -o jsonpath="{.status.loadBalancer.ingress[0].ip}" --namespace=production services gceme-frontend)

1. Confirm that both services are working by opening the frontend external IP address in your browser.
2. Check the version output of the service by running the following command (it should read 1.0.0):

curl http://$FRONTEND\_SERVICE\_IP/version

You have successfully deployed the sample application! Next, you will set up a pipeline for deploying your changes continuously and reliably.

## Task 8. Creating the Jenkins Pipeline

### **Creating a repository to host the sample app source code**

1. Create a copy of the gceme sample app and push it to a [Cloud Source Repository](https://cloud.google.com/source-repositories/docs/):

gcloud source repos create default

1. Initialize the sample-app directory as its own Git repository:

git config credential.helper gcloud.sh

1. Run the following command:

git remote add origin https://source.developers.google.com/p/$DEVSHELL\_PROJECT\_ID/r/default

1. Set the username and email address for your Git commits. Replace [EMAIL\_ADDRESS] with your Git email address and [USERNAME] with your Git username:

git config --global user.email "[EMAIL\_ADDRESS]"

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

1. Add, commit, and push the files:

git add .

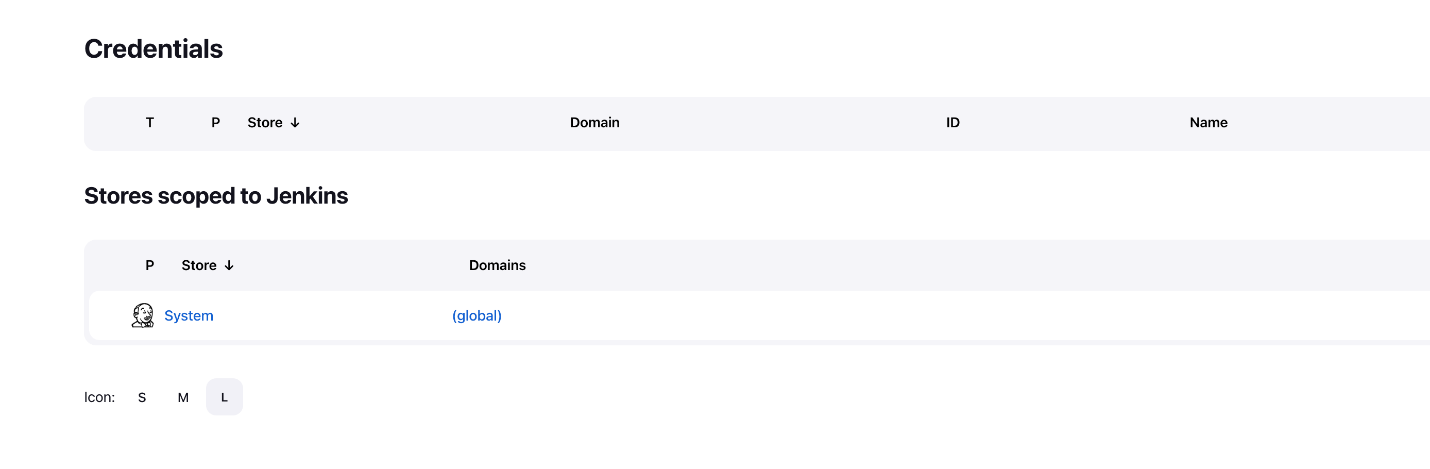
git commit -m "Initial commit"

git push origin master

### **Adding your service account credentials**

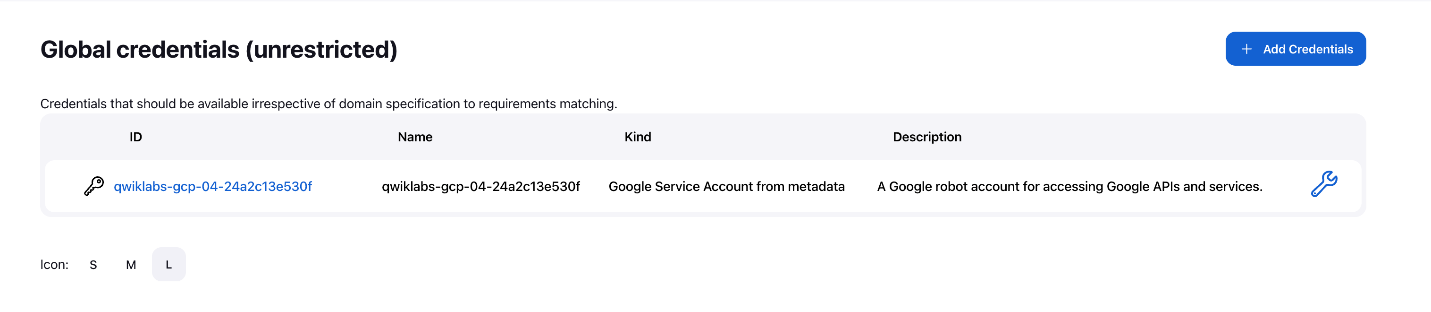
Configure your credentials to allow Jenkins to access the code repository. Jenkins will use your cluster's service account credentials in order to download code from the Cloud Source Repositories.

1. In the Jenkins user interface, click **Manage Jenkins** in the left navigation then click **Manage Credentials**.
2. Click **System**.



1. Click **Global credentials (unrestricted)**.
2. Click **Add Credentials** in the top right corner.
3. Select **Google Service Account from metadata** from the **Kind** drop-down and click **Create**.

The global credentials has been added. The name of the credential is the Project ID found in the CONNECTION DETAILS section of the lab.



### **Configure Jenkins Cloud for Kubernetes**

1. In the Jenkins user interface, select **Manage Jenkins** > **Manage nodes and clouds**.
2. Click **Configure Clouds** in the left navigation pane.
3. Click **Add a new cloud** and select **Kubernetes**.
4. Click **Kubernetes Cloud Details**.
5. In the **Jenkins URL** field, enter the following value: http://cd-jenkins:8080
6. In the **Jenkins tunnel** field, enter the following value: cd-jenkins-agent:50000
7. Click **Save**.

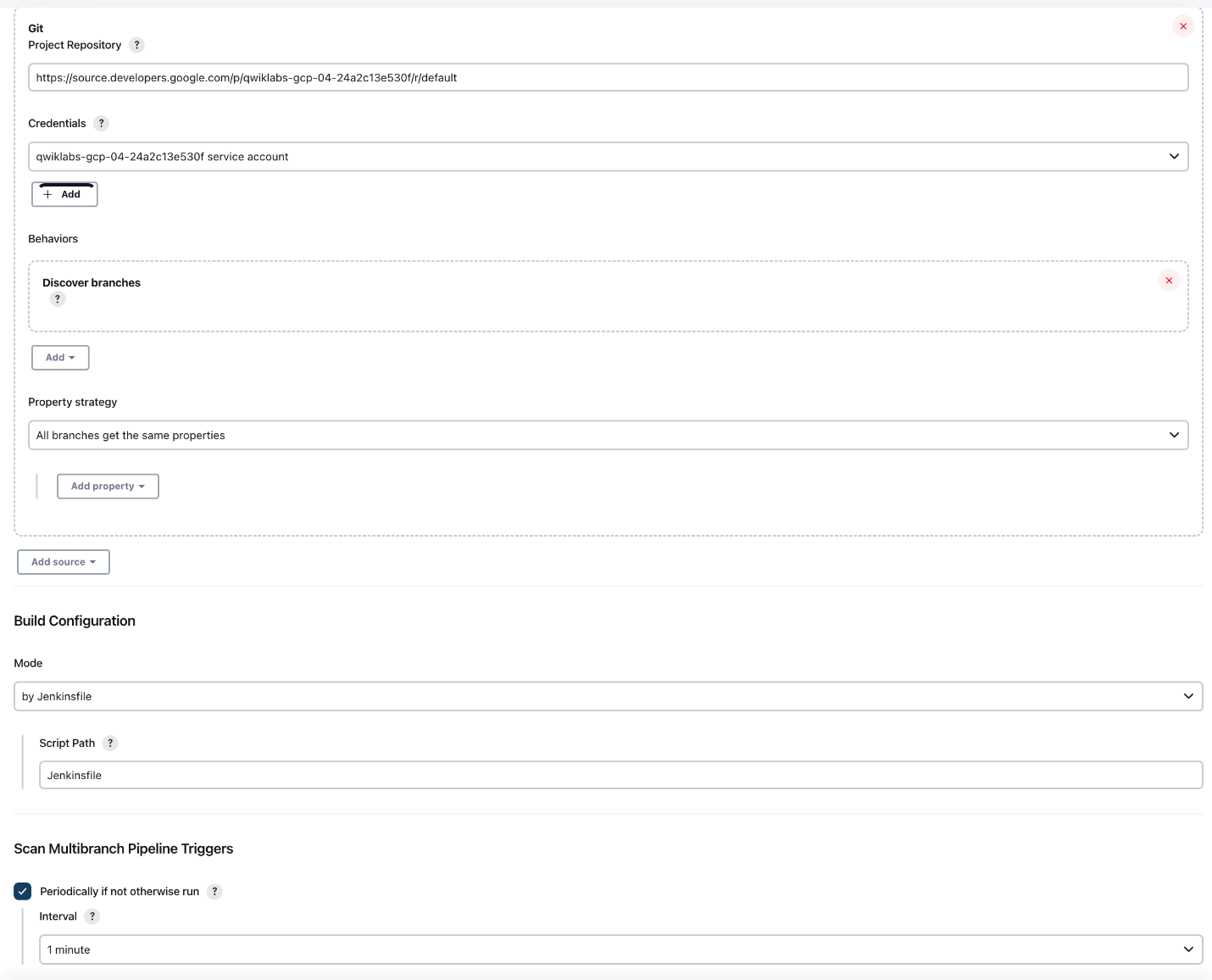
### **Creating the Jenkins job**

Navigate to your Jenkins user interface and follow these steps to configure a Pipeline job.

1. Click **Dashboard** > **New Item** in the left panel.
2. Name the project **sample-app**, then choose the **Multibranch Pipeline** option and click **OK**.
3. On the next page, in the **Branch Sources** section, select **Git** from **Add Source** dropdown.
4. Paste the **HTTPS clone URL** of your sample-app repo in Cloud Source Repositories into the **Project Repository** field. Replace [PROJECT\_ID] with your **Project ID**:

https://source.developers.google.com/p/[PROJECT\_ID]/r/default

1. From the **Credentials** drop-down, select the name of the credentials you created when adding your service account in the previous steps.
2. Under **Scan Multibranch Pipeline Triggers** section, check the **Periodically if not otherwise run** box and set the **Interval** value to **1 minute**.
3. Your job configuration should look like this:



1. Click **Save** leaving all other options with their defaults.

After you complete these steps, a job named Branch indexing runs. This meta-job identifies the branches in your repository and ensures changes haven't occurred in existing branches. If you click sample-app in the top left, the master job should be seen.

## Creating the development environment

Development branches are a set of environments your developers use to test their code changes before submitting them for integration into the live site. These environments are scaled-down versions of your application, but need to be deployed using the same mechanisms as the live environment.

### **Creating a development branch**

To create a development environment from a feature branch, you can push the branch to the Git server and let Jenkins deploy your environment.

* Create a development branch and push it to the Git server:

git checkout -b new-feature

### **Modifying the pipeline definition**

The Jenkinsfile that defines that pipeline is written using the [Jenkins Pipeline Groovy syntax](https://jenkins.io/doc/book/pipeline/syntax/). Using a Jenkinsfile allows an entire build pipeline to be expressed in a single file that lives alongside your source code. Pipelines support powerful features like parallelization and require manual user approval.

In order for the pipeline to work as expected, you need to modify the Jenkinsfile to set your project ID.

1. Open the Jenkinsfile in your terminal editor, for example vi:

vi Jenkinsfile

1. Start the editor:

i

1. Add your PROJECT\_ID to the REPLACE\_WITH\_YOUR\_PROJECT\_ID value. (Your PROJECT\_ID is your Project ID found in the CONNECTION DETAILS section of the lab. You can also run gcloud config get-value project to find it.
2. Change the value of CLUSTER\_ZONE to <filled in at lab start>. You can get this value by running gcloud config get compute/zone.
3. Save the Jenkinsfile file: hit **Esc** then (for vi users):

:wq

### **Modify the site**

To demonstrate changing the application, you will change the gceme cards from **blue** to **orange**.

1. Open html.go:

vi html.go

1. Start the editor:

i

1. Change the two instances of <div class="card blue"> with following:

<div class="card orange">

1. Save the html.go file: press **Esc** then:

:wq

1. Open main.go:

vi main.go

1. Start the editor:

i

1. The version is defined in this line:

const version string = "1.0.0"

Update it to the following:

const version string = "2.0.0"

1. Save the main.go file one more time: **Esc** then:

:wq

## Kick off Deployment

1. Commit and push your changes:

git add Jenkinsfile html.go main.go

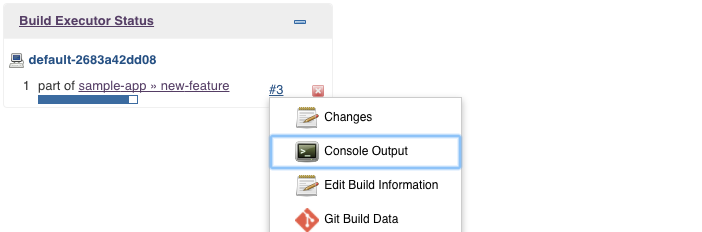
git commit -m "Version 2.0.0"

git push origin new-feature

This will kick off a build of your development environment.

After the change is pushed to the Git repository, navigate to the Jenkins user interface where you can see that your build started for the new-feature branch. It can take up to a minute for the changes to be picked up.

1. After the build is running, click the down arrow next to the **build** in the left navigation and select **Console output**:



1. Track the output of the build for a few minutes and watch for the kubectl --namespace=new-feature apply... messages to begin. Your new-feature branch will now be deployed to your cluster.

**Note:**In a development scenario, you wouldn't use a public-facing load balancer. To help secure your application, you can use [kubectl proxy](https://kubernetes.io/docs/tasks/extend-kubernetes/http-proxy-access-api/" \t "blank). The proxy authenticates itself with the Kubernetes API and proxies requests from your local machine to the service in the cluster without exposing your service to the Internet.

If you didn't see anything in Build Executor, not to worry. Just go to the Jenkins homepage > sample app. Verify that the new-feature pipeline has been created.

1. Once that's all taken care of, start the proxy in the background:

kubectl proxy &

1. If it stalls, press **Ctrl + C** to exit out. Verify that your application is accessible by sending a request to localhost and letting kubectl proxy forward it to your service:

curl \

http://localhost:8001/api/v1/namespaces/new-feature/services/gceme-frontend:80/proxy/version

You should see it respond with 2.0.0, which is the version that is now running.

If you receive a similar error:

{

"kind": "Status",

"apiVersion": "v1",

"metadata": {

},

"status": "Failure",

"message": "no endpoints available for service \"gceme-frontend:80\"",

"reason": "ServiceUnavailable",

"code": 503

1. It means your frontend endpoint hasn't propagated yet—wait a little bit and try the curl command again. Move on when you get the following output:

2.0.0

You have set up the development environment! Next, you will build on what you learned in the previous module by deploying a canary release to test out a new feature.

## Deploying a canary release

You have verified that your app is running the latest code in the development environment, so now deploy that code to the canary environment.

1. Create a canary branch and push it to the Git server:

git checkout -b canary

git push origin canary

1. In Jenkins, you should see the **canary** pipeline has kicked off. Once complete, you can check the service URL to ensure that some of the traffic is being served by your new version. You should see about 1 in 5 requests (in no particular order) returning version 2.0.0.

export FRONTEND\_SERVICE\_IP=$(kubectl get -o \

jsonpath="{.status.loadBalancer.ingress[0].ip}" --namespace=production services gceme-frontend)

while true; do curl http://$FRONTEND\_SERVICE\_IP/version; sleep 1; done

1. If you keep seeing 1.0.0, try running the above commands again. Once you've verified that the above works, end the command with **Ctrl + C**.

That's it! You have deployed a canary release. Next you will deploy the new version to production.

## Deploying to production

Now that our canary release was successful and we haven't heard any customer complaints, deploy to the rest of your production fleet.

1. Create a canary branch and push it to the Git server:

git checkout master

git merge canary

git push origin master

In Jenkins, you should see the master pipeline has kicked off.

1. Once complete (which may take a few minutes), you can check the service URL to ensure that all of the traffic is being served by your new version, 2.0.0.

export FRONTEND\_SERVICE\_IP=$(kubectl get -o \

jsonpath="{.status.loadBalancer.ingress[0].ip}" --namespace=production services gceme-frontend)

Copied!

content\_copy

while true; do curl http://$FRONTEND\_SERVICE\_IP/version; sleep 1; done

Copied!

content\_copy

1. Once again, if you see instances of 1.0.0 try running the above commands again. You can stop this command by pressing **Ctrl + C**.

**output:**

gcpstaging9854\_student@qwiklabs-gcp-df93aba9e6ea114a:~/continuous-deployment-on-kubernetes/sample-app$ while true; do curl http://$FRONTEND\_SERVICE\_IP/version; sleep 1; done

2.0.0

2.0.0

2.0.0

2.0.0

2.0.0

2.0.0

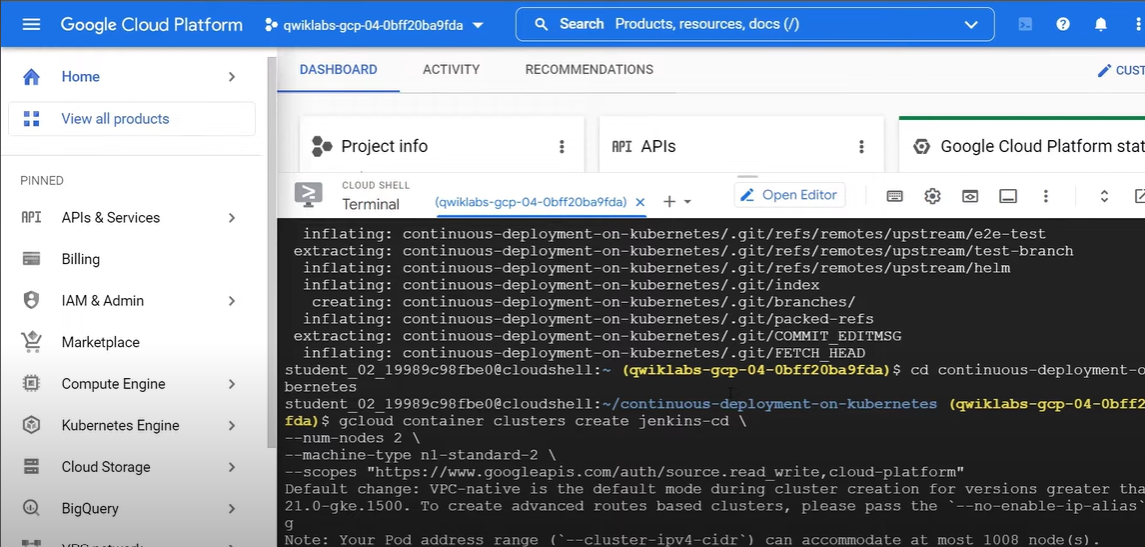
^C

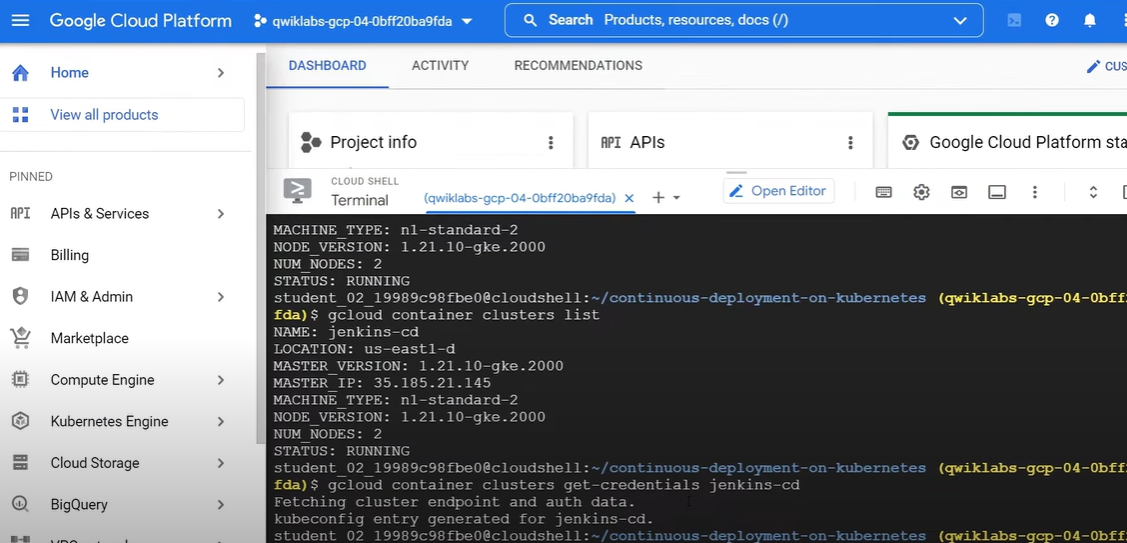
You can also navigate to site on which the gceme application displays the info cards. The card color changed from blue to orange.

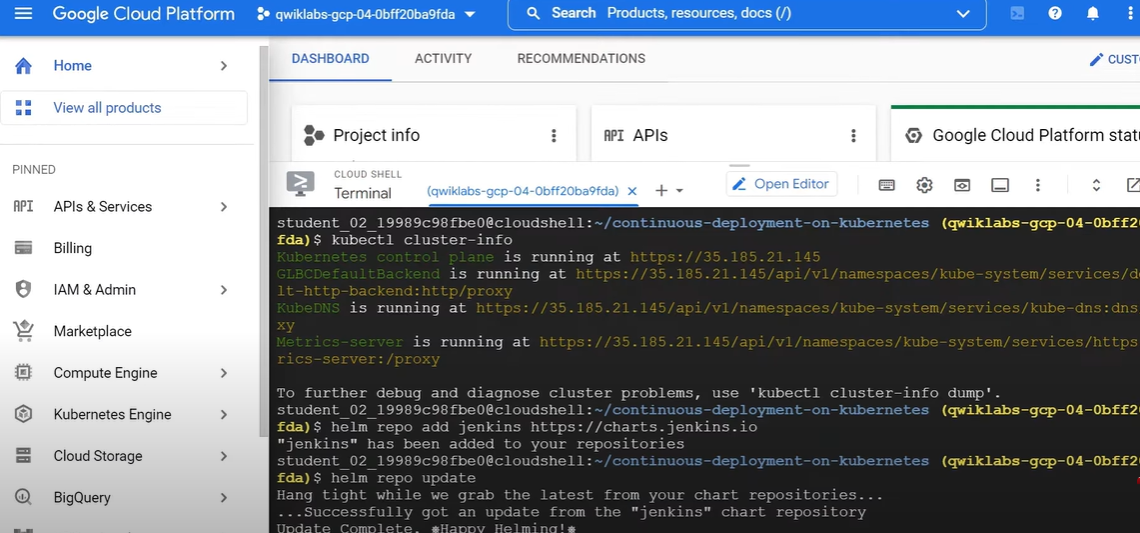
1. Here's the command again to get the external IP address. Paste the **External IP** into a new tab to see the info card displayed:

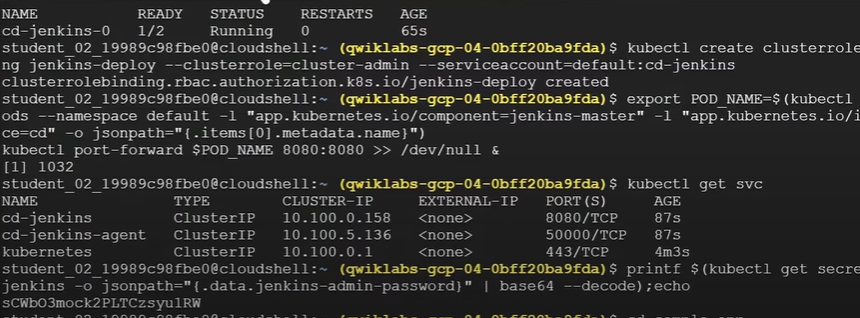
kubectl get service gceme-frontend -n production

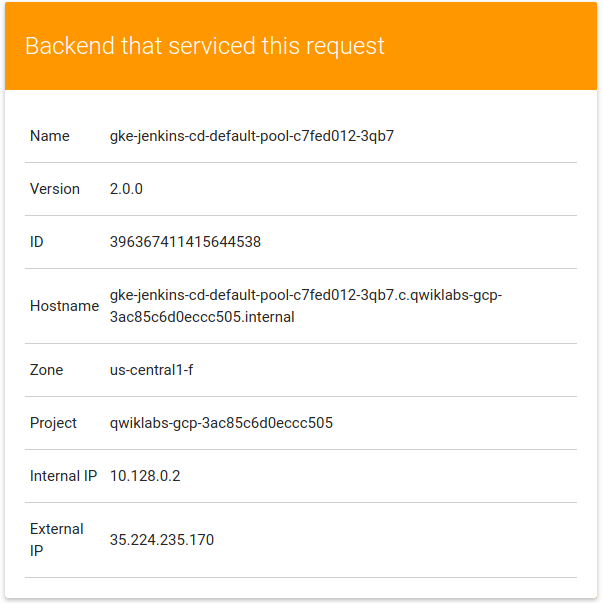
**output:**











**CONCLUSION**

Hence we know how applications are built and deployed by developers as a result of the ever-increasing demands from end users. " Cloud with Kubernetes" is a book that provides an in-depth overview of Kubernetes, an open-source container and set up a continuous delivery pipeline with Jenkins on Kubernetes engine. Jenkins is the go-to automation server used by developers who frequently integrate their code in a shared repository.

**REFERENCES**

Google Docs

<https://kubernetes.io/>

Google Cloud Boost

Aws docs

<https://docs.aws.amazon.com/>

CloudGuru

<https://acloudguru.com/>

Medium

<https://medium.com/>