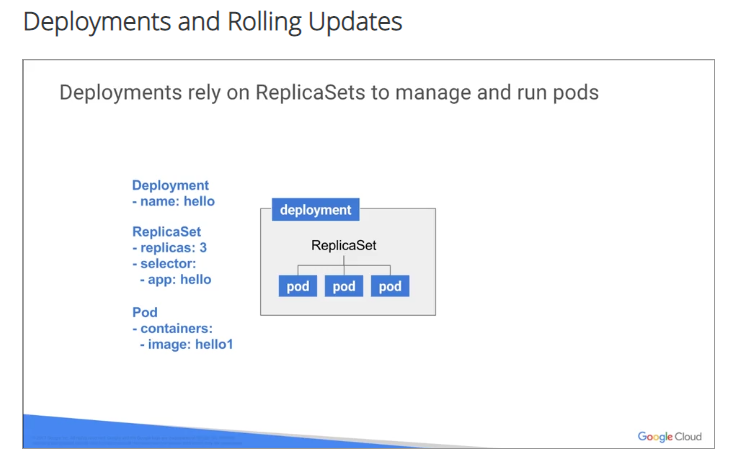
Deployments and Rolling Updates

Part of the advantage of container based applications is the ability to break them up and connect them as services.

Another is to keep development, staging, and production as similar as possible, so you can migrate between them with little or no change. For this Kubernetes adds tools called deployments and replica sets that allow you to separate and manage your build and run environments and scale them as needed. With these tools you can easily roll out common deployment such as rolling updates, canaries, and blue- greens. Now, let's take a look at what you can do.

Deploying Kubernetes and rolling updates. Deployments really rely on ReplicaSets to manage and run the pod. Deployments allow you to basically name a set of pods to ensure that the number and the state of the pods run and in an equal manner and desired number of states of pods that you have indicated within your YAML file or your configuration itself.

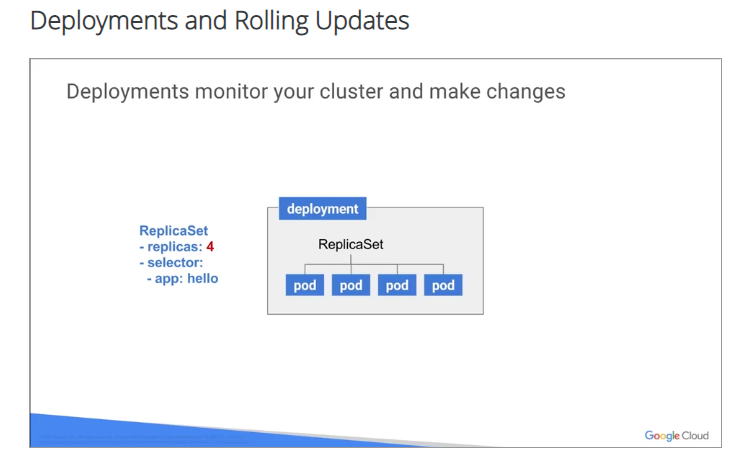
Behind the scene, what happens is the deployment relies on these ReplicaSets to manage and run any given number of pods at any given time. In this example that you see right here,there's a deployment that says, hello. When you create the deployment, it's going to create a ReplicaSet, equal size 3. When you add the label selector on there, app equals hello, it'll say inside the pod,



you have a single image and you call it hello1, simple example. As the deployment is monitored, the

cluster, you'll see whether anything is going on, whether it's different and how it's defined. And if there's anything that's different, the deployment tries to rectify it. In this example on the left-hand side,

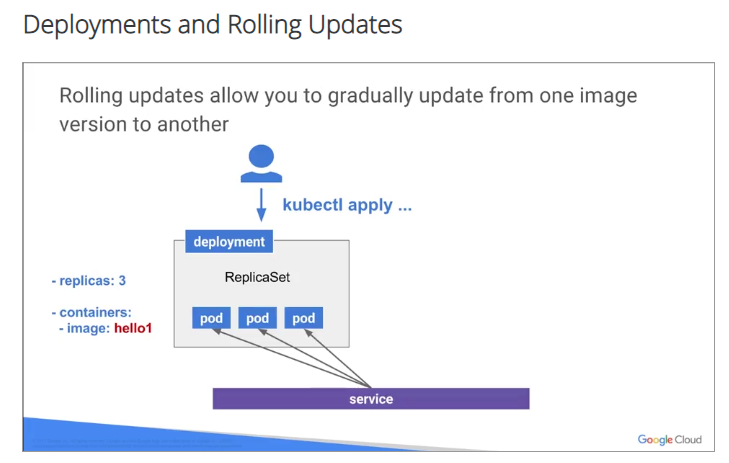
you'll see that it says ReplicaSet, 4 replicas, selector, and the application again, hello.

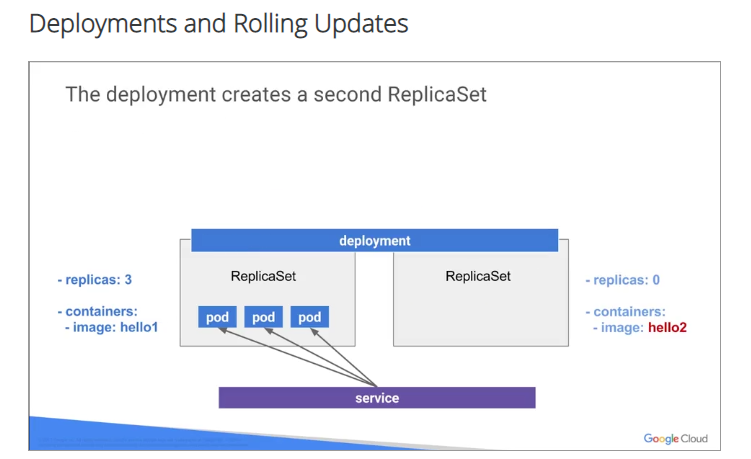


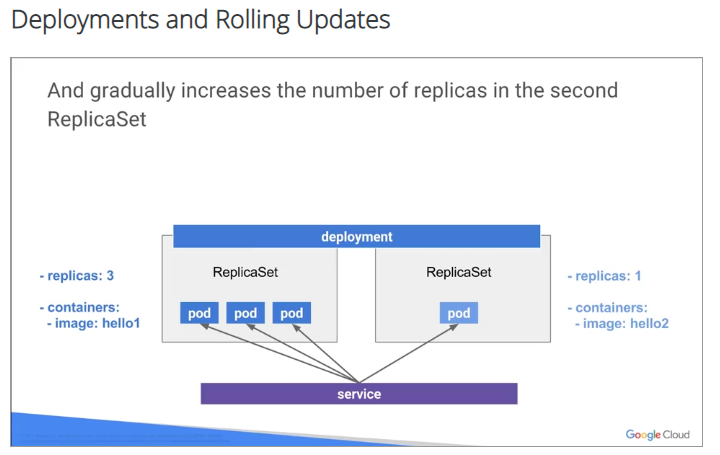
For the example, it creates a deployment with replicas of 4, like it says there on the left hand side. Only three replicas will be running. The deployment objects and the differences between them will be defined in the API. And what's running in the cluster will be defined and rectified if it needs to by running another pod somewhere else in the same cluster. If you have three pods and one of them goes down or for some reason something happens, remember we talked about being unhealthy, okay? Then the node went down, and there was a problem with the node. It upgrades it, and takes it down by the system. And then finally, another pod will be generated and rebuilt somewhere else with the original state, the original segment. So it will keep three pods running, or three replicas, better yet, running at all times as you can see. Again, rolling updates is one of the really, really awesome things, and developers love this, okay?

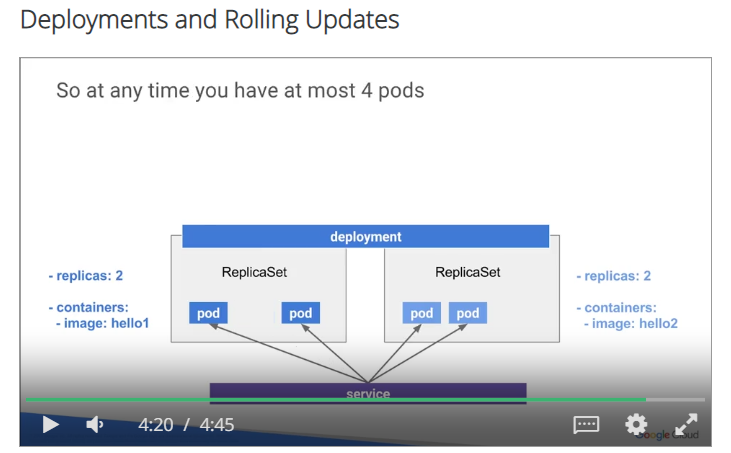
# Rolling updates

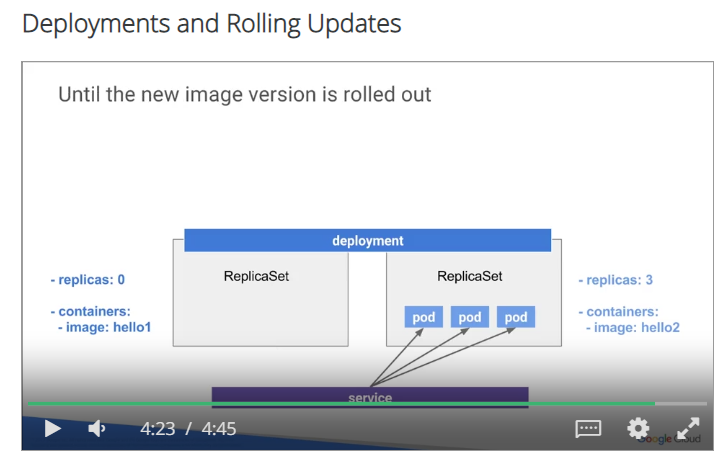
Allow you to granularly update and gradually update one image version to another image of the version without any disruption to production.

Deployment really rolls out and triggers only if it meets the deployment and it meets the pod template. And then yes, it changes the version. For example, if you have labels or containers images, templates and updates, the update will roll over and scale the deployment. And you don't need to trigger any roll out, it'll do it basically on its own









So we're going to use kubectl itself to command and apply these changes to the pods. And you're going to give the pod two versions to the images, so the pods as you could see, you have one ReplicaSet on one side. They're both, one left hand side is hello1, right hand side is hello2. And then the deployment is going to create a second ReplicaSet, like I said, hello2, and then create the pods and the second ReplicaSet. As it shuts down the pods in the old ReplicaSet, it'll keep on doing the same thing to the new ReplicaSet. And now we have a completely new version of an application rolled out, it did this during production. All the APIs were requested and changed,

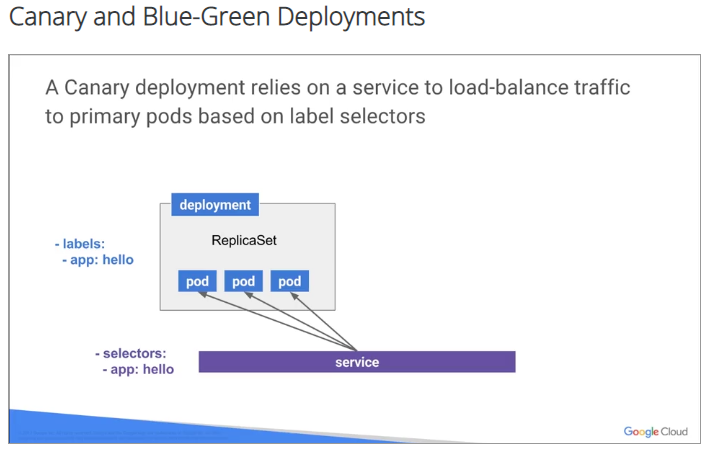
and the deployment now is sitting on version two instead of version one, and no service interruption again. Nobody knew about it. It really happens under the covers, very well done.

# Canary and Blue-green deployments

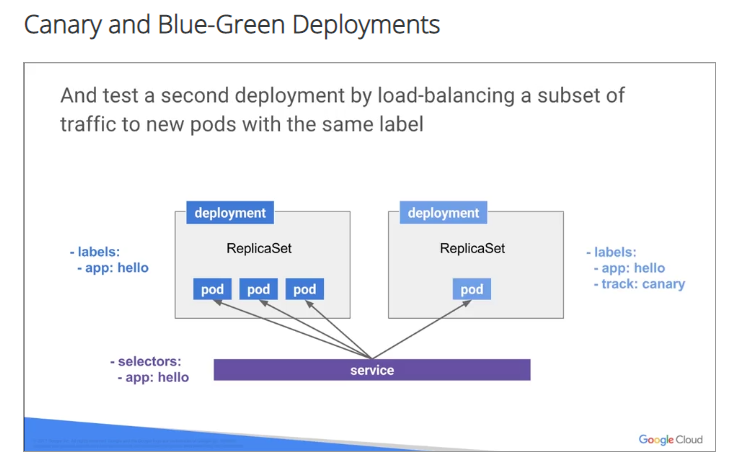
Part 2: Canary and Blue-Green deployments.

A Canary deployment relies on a service to load-balance the traffic to primarily pods based

on a label selector. Remember we said labels were not only used for finding stuff but also from the mechanism within Kubernetes.

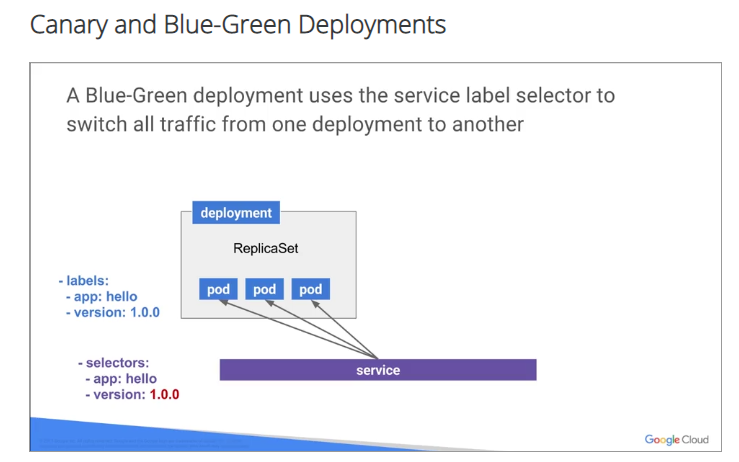


In this case you have the services pointing to a deployment that has the same services running on the label called app: hello, It's going to load-balance anything that comes in here, anything to the pods within that application with that label.

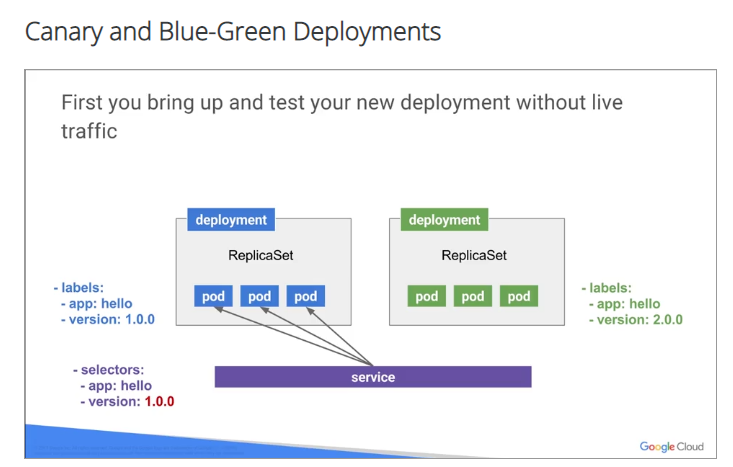


Once you deploy, you have a second deployment that also adds a second label. And you could call this on track: canary, In this case, since it's a second deployment, it also has an app: hello on it. And the services also has a load-balance against those pods. You can also try a new version of your application against a smaller subset of your live production environment while using the canary deployment also. And then if you're satisfied with all this and everything looks good, you could roll out the new deployment.

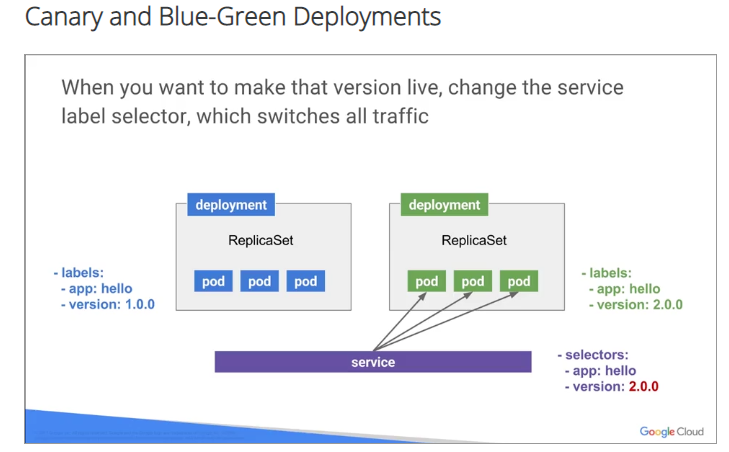
Blue-Green deployment switches all the traffic from one deployment to the other, again, no interruption.



So in this case, you have one deployment with pod labelled app: hello, version 1.0.0. And the service has all the traffic to these pods and then, now you have the new, full new deployment with the new version on it. We're going to roll it out. It's going to be full, second deployment with labels, and the label's going to be app: hello, app version: 2.0.0. So we've upgraded to the app version.



As we verified the deployment, we make sure that everything inside is running the way it needs to be running, that all the pods are in a healthy states.



Finally, what we do is we transfer the services, as soon as we know that everything is running perfectly. And then you are a back up on a new version of the application. It switches all the traffic over to the new pod. Again, no service interruption, nobody sees any interruption in the service and just keeps on going. In this lab, you'll experiment with deployment obvious and build the three types of deployments discussed in the presentation.

# Lab

# Deploying to Kubernetes v1.6

1 hour 30 minutes1 Credit

Rate Lab

## Overview

The new goal of this lab is to get you ready for scaling and managing containers in production.

And that's where [deployments](http://kubernetes.io/docs/user-guide/deployments/) come in. Deployments are a declarative way to ensure that the number of pods running is equal to the desired number of pods specified by the user.

### Introduction to Deployments

Deployments abstract away the low level details of managing pods. They provide a single stable name that you can use to update an application. Behind the scenes, deployments rely on [ReplicaSets](https://kubernetes.io/docs/user-guide/replicasets/) to manage starting, stopping, scaling, and restarting the pods if they happen to go down for some reason. If pods need to be updated or scaled, the deployment will handle all of the details for you.

Deployments (and ReplicaSets) are powered by control loops. Control loops are a design pattern for distributed software that allows you to declaratively define your desired state and have the software implement the desired state for you based on the current state. You'll see more about how that works below.

## Setup

#### Step 1

#### What you'll need

To complete this lab, you’ll need:

* Access to a standard internet browser (Chrome browser recommended).
* Time. Note the lab’s **Completion** time in Qwiklabs. This is an estimate of the time it should take to complete all steps. Plan your schedule so you have time to complete the lab. Once you start the lab, you will not be able to pause and return later (you begin at step 1 every time you start a lab).
* The lab's **Access** time is how long your lab resources will be available. If you finish your lab with access time still available, you will be able to explore the Google Cloud Platform or work on any section of the lab that was marked "if you have time". Once the Access time runs out, your lab will end and all resources will terminate.
* You **DO NOT** need a Google Cloud Platform account or project. An account, project and associated resources are provided to you as part of this lab.
* If you already have your own GCP account, make sure you do not use it for this lab.
* If your lab prompts you to log into the console, **use only the student account provided to you by the lab**. This prevents you from incurring charges for lab activities in your personal GCP account.

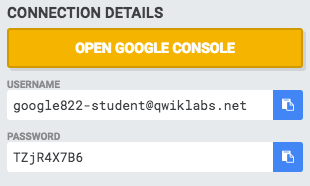
#### Start your lab

When you are ready, click **Start Lab**. You can track your lab’s progress with the status bar at the top of your screen.

**Important** What is happening during this time? Your lab is spinning up GCP resources for you behind the scenes, including an account, a project, resources within the project, and permission for you to control the resources needed to run the lab. This means that instead of spending time manually setting up a project and building resources from scratch as part of your lab, you can begin learning more quickly.

#### Find Your Lab’s GCP Username and Password

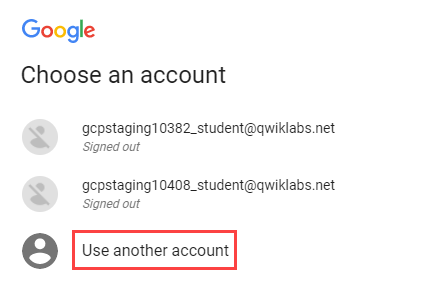
To access the resources and console for this lab, locate the Connection Details panel in Qwiklabs. Here you will find the account ID and password for the account you will use to log in to the Google Cloud Platform:



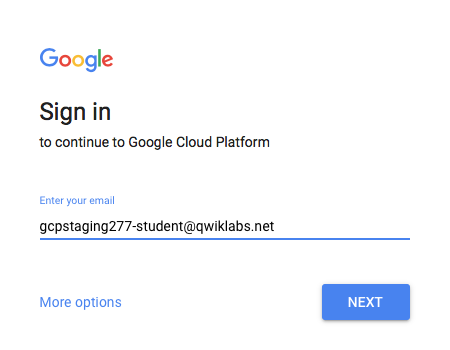
If your lab provides other resource identifiers or connection-related information, it will appear on this panel as well.

#### Log in to Google Cloud Console

Using the Qwiklabs browser tab/window or the separate browser you are using for the Qwiklabs session, copy the Username from the Connection Details panel and click the **Open Google Console** button.

You'll be asked to Choose an account. Click **Use another account**. 

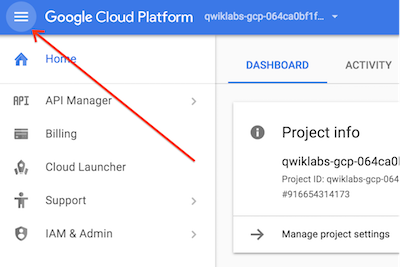
Paste in the Username, and then the Password as prompted:



Accept the terms and conditions.

Since this is a temporary account, which you will only have to access for this one lab:

* Do not add recovery options
* Do not sign up for free trials

**Note:** You can view the list of services by clicking the GCP Navigation menu button at the top-left next to “Google Cloud Platform”.

#### Step 2

Make sure the following APIs are enabled in Cloud Platform Console:

* Kubernetes Engine API
* Container Registry API

On the **Navigation menu** (Navigation menu), click **APIs & services**.

Scroll down and confirm that your APIs are enabled.

If an API is missing, click **ENABLE APIS AND SERVICES** at the top, search for the API by name, and enable it for your project.

#### Step 3

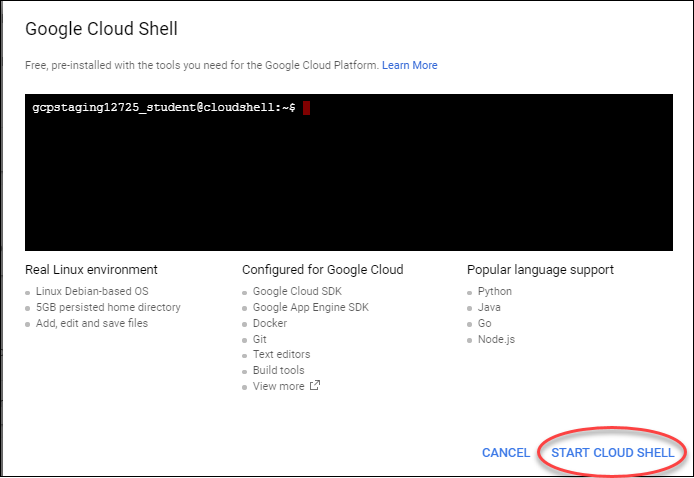
### Activate Google Cloud Shell

Google Cloud Shell provides command-line access to your GCP resources.

From the GCP Console click the **Cloud Shell** icon on the top right toolbar:

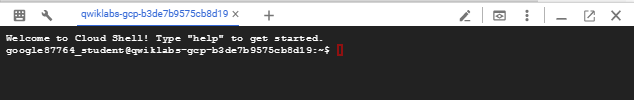


Then click **START CLOUD SHELL**:



You can click **START CLOUD SHELL** immediately when the dialog comes up instead of waiting in the dialog until the Cloud Shell provisions.

It takes a few moments to provision and connects to the environment:



The Cloud Shell is a virtual machine loaded with all the development tools you’ll need. It offers a persistent 5GB home directory, and runs on the Google Cloud, greatly enhancing network performance and authentication.

Once connected to the cloud shell, you'll see that you are already authenticated and the project is set to your PROJECT\_ID:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

**Note:** gcloud is the powerful and unified command-line tool for Google Cloud Platform. Full documentation is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud). It comes pre-installed on Cloud Shell and supports tab-completion.

gcloud config list project

Output:

[core]

project = <PROJECT\_ID>

#### Step 4

Define your zone as a project default zone. This way you do not need to specify --zone parameter in gcloud commands.

gcloud config set compute/zone us-central1-a

Get the sample code for creating and running containers and deployments:

git clone https://github.com/googlecodelabs/orchestrate-with-kubernetes.git

#### Step 5

Start your Kubernetes cluster with 5 nodes.

cd orchestrate-with-kubernetes/kubernetes

In Cloud Shell, run the following command to start a Kubernetes cluster called bootcamp that runs 5 nodes.

gcloud container clusters create bootcamp --num-nodes 5 --scopes "https://www.googleapis.com/auth/projecthosting,storage-rw"

The scopes argument provides access to project hosting and Google Cloud Storage APIs that you'll use later.

It takes several minutes to create a cluster as Kubernetes Engine provisions virtual machines for you. It spins up one or more master nodes and multiple configured worker nodes. This is one of the advantages of a managed service.

## Learn About Deployment Objects

#### Step 1

Run the explain command in kubectl to tell you about the deployment object.

kubectl explain deployment

#### Step 2

Run the command with the --recursive option to see all of the fields.

kubectl explain deployment --recursive

#### Step 3

Use the explain command as you go through the lab to help you understand the structure of a deployment object and understand what the individual fields do.

kubectl explain deployment.metadata.name

### Create a Deployment

Create a simple deployment.

#### Step 1

Examine the deployment configuration file.

cat deployments/auth.yaml

kubectl create will create the auth deployment with one replica, using version 1.0.0 of the auth container. To scale the number of pods, you simply change the replicas field.

#### Step 2

Create the deployment object using kubectl create.

kubectl create -f deployments/auth.yaml

#### Step 3

Verify that it was created.

kubectl get deployments

#### Step 4

Kubernetes creates a ReplicaSet for the deployment.

Run the following command to verify it. You should see a ReplicaSet with a name like auth-xxxxxxx.

kubectl get replicasets

#### Step 5

Run the following command to view the pods created for your deployment. A single pod was created when the ReplicaSet was created.

kubectl get pods

#### Step 6

With your pod running, it's time to put it behind a service. Use the kubectl create command to create the auth service.

kubectl create -f services/auth.yaml

#### Step 7

Do the same to create and expose the hello and frontenddeployments.

kubectl create -f deployments/hello.yaml

kubectl create -f services/hello.yaml

kubectl create configmap nginx-frontend-conf --from-file=nginx/frontend.conf

kubectl create secret generic tls-certs --from-file tls/

kubectl create -f deployments/frontend.yaml

kubectl create -f services/frontend.yaml

You created a ConfigMap and secret for the frontend.

Click Check my progress to verify the objective.

Create cluster and deployments (Auth, Hello, and Frontend)

Check my progress

#### Step 8

Interact with the frontend.

Get its external IP.

kubectl get services frontend

You may need to re-run this command every few seconds until the External IP is populated.

And curl the service.

curl -ks https://<EXTERNAL-IP>

You get the "hello" response. Use the output templating feature of kubectl to run curl as a one-line command.

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`

### Scale a Deployment

Update the spec.replicas field to scale the deployment.

#### Step 1

Run the kubectl explain command to see an explanation of the field.

kubectl explain deployment.spec.replicas

#### Step 2

You can update the replicas field most easily using the kubectl scale command.

kubectl scale deployment hello --replicas=5

It may take a minute or so for all the new pods to start up.

#### Step 3

Kubernetes updates the ReplicaSet and starts new pods to equal 5.

Verify there are 5 pods running.

kubectl get pods | grep hello- | wc -l

#### Step 4

Scale back the application.

kubectl scale deployment hello --replicas=3

#### Step 5

Verify the correct number of pods.

kubectl get pods | grep hello- | wc -l

**Congratulations!**

You learned about Kubernetes deployments and how to manage and scale a group of pods.

## Rolling Updates

Deployments update images to new versions through rolling updates. When a deployment is updated with a new version, it creates a new ReplicaSet and slowly increases the number of replicas in the new ReplicaSet as it decreases the replicas in the old ReplicaSet.

### Trigger a Rolling Update

#### Step 1

Run the following command to update your deployment.

kubectl edit deployment hello

#### Step 2

Change the image in containers section to the following, then save and exit.

containers:

- name: hello

image: kelseyhightower/hello:2.0.0

The editor uses **vi** commands:

1. Use arrow keys to hover over version number **1**
2. Type **r** to replace it, and enter **2**
3. Type **:wq!** and hit **Enter** to write and quit the file.

If you have difficulty and are in a class, ask your instructor for help.

The updated deployment is saved to your cluster and Kubernetes begins a rolling update.

#### Step 3

You can see the new ReplicaSet that Kubernetes creates.

kubectl get replicaset

If you fail to see a new ReplicaSet, make sure you changed the image in containers, and not one of the other references in labels.

#### Step 4

View the new entry in the rollout history.

kubectl rollout history deployment/hello

### Pause a Rolling Update

If you detect problems with a running rollout, pause it to stop the update.

#### Step 1

Pause the update.

kubectl rollout pause deployment/hello

#### Step 2

Verify the current state of the rollout.

kubectl rollout status deployment/hello

#### Step 3

Verify this with the pods.

kubectl get pods -o jsonpath --template='{range .items[\*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'

### Resume a Rolling Update

The rollout is paused which means that some pods are at the new version and some pods are at the older version.

#### Step 1

Use the resume command to continue the rollout.

kubectl rollout resume deployment/hello

#### Step 2

Run the status command to verify the rollout is complete.

kubectl rollout status deployment/hello

You'll get the following:

deployment "hello" successfully rolled out

### Rollback an Update

If a bug occurs in your new version, users connected to new pods will experience the issue.

#### Step 1

Use the rollout undo command to roll back to the previous version, then fix any bugs.

kubectl rollout undo deployment/hello

#### Step 2

Verify the rollback in the deployment's history.

kubectl rollout history deployment/hello

#### Step 3

Verify all pods have rolled back to the previous version.

kubectl get pods -o jsonpath --template='{range .items[\*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'

**Congratulations!**

You learned how to roll out application updates without downtime.

## Canary Deployments

Run a canary deployment to test a new deployment in production with a subset of users. This mitigates risk with new releases.

### Create a Canary Deployment

A canary deployment consists of a separate deployment from your stable deployment and a service that targets them both at the same time.

#### Step 1

Examine the file that creates a canary deployment for your new version.

cat deployments/hello-canary.yaml

It includes the following:

* the deployment hello-canary
* 1 pod (replica)
* selectors app: hello and track: canary
* an image with version 2.0.0.

#### Step 2

Create the canary deployment.

kubectl create -f deployments/hello-canary.yaml

Click Check my progress to verify the objective.

Canary Deployments

Check my progress

#### Step 3

After the canary deployment is created, verify you have two deployments hello and hello-canary.

kubectl get deployments

The hello service selector uses app: hello, which matches pods in both deployments. However, the canary deployment has fewer pods, and is only used by a subset of users.

### Verify the Canary Deployment

You can verify both hello versions being served by requests.

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Run the command several times and confirm that hello 1.0.0 serves about ¾ (75%) of requests and 2.0.0 serves about ¼ (25%).

By default, every request has a chance to be served by the canary deployment. If you want users to get all their responses from the same version, enable session affinity in the configuration file as follows:

spec:

sessionAffinity: ClientIP

### Clean Up

You're done using the canary deployment.

Delete it and the service as follows.

kubectl delete deployment hello-canary

**Congratulations!**

You learned about canary deployments and how to test new versions of an application in a live environment.

## Blue-Green Deployments

You can use blue-green deployments if it's more beneficial to modify load balancers to point to a new, fully-tested deployment all at once.

A downside is you need double the resources to host both versions of your application during the switch.

### The Service

You use the existing hello deployment for the blue version and a newhello-green deployment for the green version.

Deployments have the following label:

|  |  |  |
| --- | --- | --- |
| Deployment | Label Name | Label Value |
| hello (blue) | version | 1.0.0 |
| hello-green | version | 2.0.0 |

You use two nearly-identical service files (hello-blue and hello-green) to switch between versions. The only difference between these files is their version selector. You could edit the service while it's running and change the version selector, but switching files is easier for labs.

First, update the service to use the blue deployment:

kubectl apply -f services/hello-blue.yaml

### Create a Blue-Green Deployment

#### Step 1

Create the green deployment.

kubectl create -f deployments/hello-green.yaml

#### Step 2

Verify the blue deployment (1.0.0) is still being used.

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

#### Step 3

Run the following command to update the service to use the green deployment.

kubectl apply -f services/hello-green.yaml

#### Step 4

Verify the green deployment is being used.

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

### Rollback a Blue-Green Deployment

You can roll back to the old version.

#### Step 1

While the green deployment is still running, simply update the service to the old (blue) deployment.

kubectl apply -f services/hello-blue.yaml

#### Step 2

Verify that the blue deployment is being used.

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

**Congratulations!**

You learned how to use blue-green deployments to switch application versions all at once.

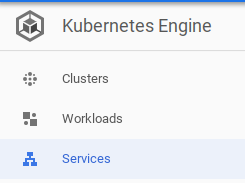
## Use the Web UI

You can also use Admin Console to view and interact with your Kubernetes Engine objects in the web interface rather than the command-line.

#### Step 1

In Admin Console, click **Navigation menu** and select **Kubernetes Engine**.

Select a resource in the tab on the left, for example **Services**.

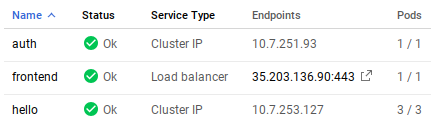


#### Step 2

Investigate each set of objects by navigating through the following pages:

* **Clusters**
* **Workloads** (Pods)
* **Services**
* **Configuration** (ConfigMaps and Secrets).

On each page, scroll over an item in the **Name** field and click its link to view more information about it. For example, on the **Services** page, click **frontend**. Your front end Services will display as follows with status, types, endpoints, and number of running Pods:



## End your lab

# Quiz

1, What is purpose of the ReplicaSet? To create desired number of PODs

2, When are rolling deployments triggered? ~~User request,~~ pod template changes

3, How does Kubernetes choose instances in the second deployment of a canary deployment? The service points to instances in both deployments with a common selector