Deploying to Kubernetes

Overview

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

And that's where [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](#) 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:
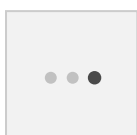
If your lab provides other resource identifiers or connection-related information, it will appear on this panel as well.
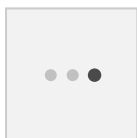
Log in to Google Cloud Console

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**.

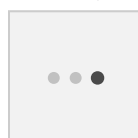Paste in the Username, and then the Password as prompted:

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** ( ), click **APIs & services**.
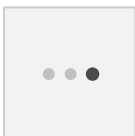
Scroll down and confirm that your APIs are enabled.

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](). 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 `frontend` deployments.

```
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.

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
```

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 new `hello-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 Cluster Dashboard [optional]

Unrelated to deployments, you can view and interact with cluster resources in a web dashboard rather than the command-line.

To use the dashboard:

- Configure `kubectl` to communicate with a cluster (already done)
- Use `kubectl` to start an HTTP proxy that connects with the Kubernetes API server
- Point a browser to the proxy to get your cluster's dashboard.

You can use Cloud Shell to do this because `kubectl` is already running on it. All you have to do is start the proxy and point a browser to the dashboard.
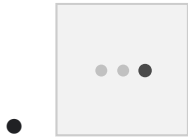
Step 1
Run the following command in Cloud Shell to start the proxy on port 8080.

```
kubectl proxy --port=8080
```

Step 2
In Cloud Shell's menu bar, click **Web Preview** > **Preview on port 8080** to open a browser pointed to Cloud Shell's port 8080.

-

Step 3

There should be a URL in place of http://localhost:8080 and 'ui' is to be appended to it (it should look like https://8080-dot-3274376-dot-devshell.appspot.com/ui). This displays the dashboard.

For more details, see https://cloud.google.com/kubernetes-engine/docs/oss-ui.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied

- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.

Last Tested Date: 2018-10-16

Last Updated Date: 2018-10-16