LAB 5 – Deploying Applications Using Google Container Engine

1. Overview

In this lab, you use create a Google Container Engine cluster and deploy the Bookshelf application to it using the Kubernetes command-line tool, kubectl. You continue to use existing storage services you used earlier in the course, including Google Cloud Datastore, and Google Cloud Storage.

What you need

To complete this lab, you need:

Access to a supported Internet browser:

* The latest version of Google Chrome, Firefox, or Microsoft Edge
* Microsoft Internet Explorer 11+
* Safari 8+ (Safari private mode is not supported)

A Google Cloud Platform project

A Source Code repository containing the lab code

A Google Cloud Storage bucket

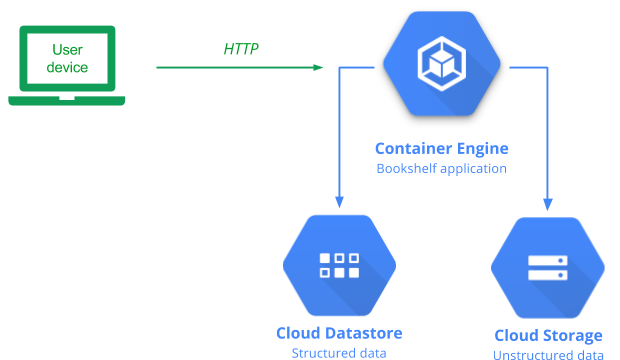
What you learn

In this lab, you:

* Create a container cluster using the Cloud SDK
* Build and push a Bookshelf image to Container Registry
* Use kubectl to deploy the Bookshelf container

2. Introduction

In this lab, you explore [Google Container Engine](https://cloud.google.com/container-engine/docs/) using the Bookshelf sample application. Container Engine is only used in this lab to host the frontend Python web component of the application. You continue to use the existing Google Cloud Platform storage services used earlier in the course - Cloud Datastore to host structured data for books, and Cloud Storage to host book cover images. The following diagram illustrates the high-level components and resources that make up the version of Bookshelf you deploy in this lab.



Container Engine is a cluster manager and orchestrator for running Docker containers. Container Engine schedules containers into a cluster and automatically manages them based on requirements you define [declaratively](https://en.wikipedia.org/wiki/Declarative_programming) using configuration files. It is built on the open source [Kubernetes](http://kubernetes.io/) system, and integrates with other Google Cloud Platform services using dedicated client libraries for a variety of supported languages.

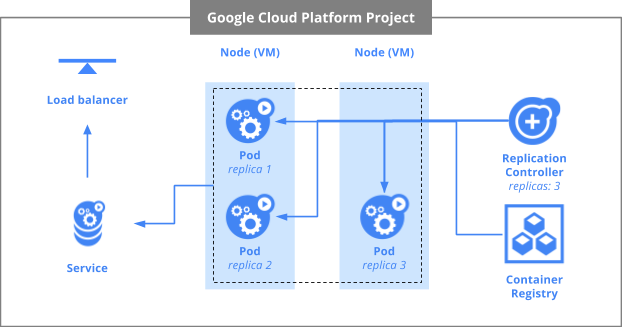
Container Engine clusters run on Compute Engine instances or virtual machines known as nodes. In this lab, you create the container cluster using the Cloud SDK and then use the [kubectl](http://kubernetes.io/docs/user-guide/kubectl-overview/" \t "_blank) command line tool to deploy and scale containers running Bookshelf on Kubernetes. One major advantage of this version of Bookshelf is that it is straightforward to deploy the code to any environment running the required version of Kubernetes - your laptop, on-premises, or even hybrid deployments across public and private clouds.

Kubernetes includes a number of concepts that are new for many developers, and that are covered in this brief introductory lab, including:

* [Nodes](http://kubernetes.io/docs/admin/node/): A node is a worker machine in a Kubernetes cluster, and in Google Container Engine, the machine is always a [Compute Engine instance](https://cloud.google.com/compute/docs/instances/). In this lab, you create a cluster with two Compute Engine nodes.
* [Pods](http://kubernetes.io/docs/user-guide/pods/): A pod is a group of one or more containers, shared storage, and configuration data relating to those containers. It is common for production applications running in Kubernetes to include multiple, relatively tightly-coupled containers in a single pod. For the purpose of this simple demonstration however, a single Bookshelf Docker container runs inside each pod.
* [Replication Controllers](http://kubernetes.io/docs/user-guide/replication-controller/): A replication controller works to ensure that the requested number of pod replicas are always available and running at a given time. The replication controller automatically adds or removes pods as required to maintain a desired state. In the context of this lab, you specify that the replication controller should create three replicas, running across your two nodes. Once you deploy the replication controller, there will be three identical pods running the frontend component of Bookshelf.
* [Services](http://kubernetes.io/docs/user-guide/services/): A service defines a logical set of pods and a way to access them using an IP address and port number pair. In the context of this lab, the Bookshelf application is exposed to end-users as a service using a Google Cloud Platform [network load-balancer](https://cloud.google.com/compute/docs/load-balancing/network/), and external IP address on port 80.

In addition to Container Engine, you also use [Google Container Registry](https://cloud.google.com/container-registry/docs/) in this lab to host a Docker image of Bookshelf. Container Registry provides secure, and private Docker image storage on Google Cloud Platform.

The resources summarized in this brief introduction are illustrated below, including how they relate to a Google Cloud Platform project. Note that this illustration is conceptually very similar to the deployment you build in this lab.



3. Create a container cluster

Enable any APIs required in this lab. Create a container cluster using the Cloud SDK. Review the configuration files for the Bookshelf application.

To create a container cluster:

Step 1

Open the [Google Cloud Platform Console](https://console.cloud.google.com/), and if necessary, select the **cp100** project.

Step 2

Unlike the App Engine standard environment, applications running on Container Engine or Compute Engine do not automatically gain access to the Cloud Datastore API or the Container Engine API. You must enable these APIs in your project for Bookshelf to work correctly in this lab.

In the Cloud Platform Console navigator, click**API Manager**.

Step 3

Click **Enable API**.

Step 4

In the **search** field type **Container Engine API**.

Step 5

Click **Google Container Engine API**.

Step 6

Click **Enable** to enable the API.

Step 7

In the top right corner of the console window, click the **Activate Google Cloud Shell** button (https://codelabs.developers.google.com/codelabs/cp100-container-engine/img/2f6b3347d4d0ebfd.png).

Step 8

You need to configure the Cloud SDK in Cloud Shell with a default zone in which to create your container cluster. Type the following command to view a list of available zones.

gcloud compute zones list

The output lists the available zones names, the region they are based in, and status.

Select a zone from the region geographically closest to your location.

Step 9

Type the following command to configure the Cloud SDK with a default zone and substitute your chosen value where indicated in the following command.

gcloud config set compute/zone <ZONE>

For example, if you chose the zone us-east1-b then the command would look as follows.

gcloud config set compute/zone us-east1-b

Step 10

Type the following command to review your current Cloud SDK configuration.

gcloud config list

The output lists a variety of settings, including the configuration you applied in the previous step under the [compute]section:

[compute]

..

zone = <ZONE>

..

Step 11

You are now ready to create a container cluster. The cluster needs access to a variety of authorization scopes to allow it access to resources in the same project such as Cloud Datastore. The simplest way for this demonstration is to grant the cluster nodes the following scopes.

* cloud-platform
* https://www.googleapis.com/auth/userinfo.email

The containers in the cluster run on a number of virtual machine nodes. In Container Enginer these nodes are provisioned as Compute Engine instances. In this lab, you use 2 nodes as the resource requirements are relatively modest and this serves to illustrates deployment to a small cluster. In a production scenario you might run a cluster with many tens, or even hundreds of nodes.

Type the following command to create a container cluster called bookshelf.

gcloud container clusters create bookshelf \

--scopes "https://www.googleapis.com/auth/userinfo.email","cloud-platform" \

--num-nodes 2

This command will take several minutes to complete as the container cluster nodes are first created, and then Kubernetes is started and configured. Once the operation is complete, the output will summarize the details of your new container cluster, including name, location, and IP address.

You can continue to the next step while this operation completes in the background.

Step 12

Before you continue on to the next section, take a few moments to look through some of the notable changes in this version of Bookshelf. Unlike the code you explored earlier in the course for the App Engine standard environment, this version makes use of alternative client libraries for both Cloud Datastore, and Cloud Storage that are more suited to Container Engine and Compute Engine.

The [Google Cloud Client Libraries](https://cloud.google.com/sdk/cloud-client-libraries) offer support for creating applications built on Google Cloud Platform services in a variety of languages, including Go, Node.js, Python, Java and Ruby.

Click **Tools > Development**.

Step 13

Click **/ > container-engine > bookshelf > model\_datastore.py**.

This code is conceptually similar to the equivalent file in the App Engine version of Bookshelf you examined earlier in the course. One notable difference, is that the functions in this version rely on the client library described in the previous step, and datastore is imported on **line 16** from gcloud. Perhaps the biggest difference is that in this version, the client library must be explicitly initialize in the get\_client() function on **lines 26 through 27**.

Step 14

Click **/ > container-engine > bookshelf > storage.py**.

This file has also been updated to make use of the Google Cloud Client Library for Python, imported on **line 18**. Notice again that the client is initialized in the \_get\_storage\_client() function on **lines 24 through 26**, before it can be used in the upload\_file() function on **lines 49 through 70** to interact with the service.

Step 15

Click **/ > container-engine > bookshelf-frontend.yaml**.

This [YAML](https://en.wikipedia.org/wiki/YAML) configuration file defines both a Kubernetes replication controller on **lines 24 through 52**, and Kubernetes service on **lines 63 through 77**.

Notice on **line 29** that the replication controller is configured to ensure 3 replicas of the bookshelf image are running at all times. The source of the image is defined on **line 39**. Notice also that the containers are configured on **line 52** for access on port 8080.

The Kubernetes service provides a load-balancing proxy for the bookshelf application. By specifying the type as a LoadBalancer on **line 71**, Container Engine creates an external HTTP load balancer in Google Cloud Platform. **Lines 72 through 74** expose the internal containers running on port 8080, externally on port 80.

Step 16

Leave the Cloud Platform Console and Cloud Shell windows open.

## 4. Deploy Bookshelf

Build and push the image of Bookshelf to Container Registry. Use the kubectl command utility to deploy the Bookshelf image on your container cluster.

To deploy bookshelf:

## Step 1

Return to Cloud Shell and check that the command you issued in the previous section to create a container cluster called bookshelf has completed before proceeding.

## Step 2

Type the following command to change to the directory containing the code for this lab.

cd ~/cp100/default/container-engine

## Step 3

Type the following command to replace the project ID and Cloud Storage bucket name placeholders in config.py with your project ID.

**Note**

Note that this one command configures the file with both values. Also note that "your-project-id" is the text being replaced by the command.

sed -i s/your-project-id/$DEVSHELL\_PROJECT\_ID/ config.py

There is no output for this command.

## Step 4

You also need to update the Kubernetes YAML configuration file you reviewed earlier, bookshelf-frontend.yaml,with your project ID.

Type the following command to replace "your-project-id" in bookshelf-frontend.yaml, with your project ID.

sed -i s/your-project-id/$DEVSHELL\_PROJECT\_ID/ bookshelf-frontend.yaml

There is no output for this command.

## Step 5

The Cloud SDK can be configured to use a container cluster as the default value for relevant commands.

Type the following command to configure bookshelf as the default container cluster for your Cloud SDK.

gcloud config set container/cluster bookshelf

## Step 6

Before you can deploy the Bookshelf application to Container Engine, you must build a Docker image and push it to Container Registry. Your bookshelf container cluster will then pull the container image from the registry when you deploy it. Type the following command to build the Docker image and tag it with the URL for Container Registry in your project.

**Note**

Recall that $DEVSHELL\_PROJECT\_ID is a shell variable that returns the value of the project ID from where Cloud Shell was started.

docker build -t gcr.io/$DEVSHELL\_PROJECT\_ID/bookshelf .

The output displays detailed information about the build process. This will take a few minutes to complete and once the process is finished you should see output similar to the following.

Successfully built <###>

Recall that in the case of the App Engine version of the application, you had to manually install various Python packages listed in the requirements.txt file. This version of the application is built using the Docker command you run in this step. The requirements.txt file is incorporated into the build process so you do not need to perform this step separately.

## Step 7

You can now push the built image to Container Registry using the Cloud SDK. Your container cluster will then download the Bookshelf image from Container Registry. Type the following command to push your new image.

gcloud docker -- push gcr.io/$DEVSHELL\_PROJECT\_ID/bookshelf

The output displays the progress of the push to Container Registry. The operation may take several minutes to complete.

## Step 8

You use the kubectl command utility to manage resources in your Container Engine cluster. To use the utility you must first configure it with credentials. Type the following command to retrieve the credentials using the Cloud SDK.

gcloud container clusters get-credentials bookshelf

The output confirms that the credentials have been fetched.

Fetching cluster endpoint and auth data.

kubeconfig entry generated for bookshelf.

## Step 9

Now that the container image of Bookshelf is built and hosted in Container Registry, you can deploy it to your container cluster. Kubernetes provides a declarative API and you simply need to instruct your container cluster to use your configuration file to deploy the Bookshelf application.

Recall that you edited the YAML configuration file earlier in the lab to include the address for Container Registry in your project. Type the following kubectl command to deploy the Bookshelf application.

kubectl create -f bookshelf-frontend.yaml

The output confirms that the "bookshelf-frontend" replication controller and service have been created.

## Step 10

Type the following kubectl command to check the status of your pods.

kubectl get pods

You should see your Bookshelf pods listed and the status should be ‘Running.' It may take a moment for the status to change from ‘ContainerCreating' to ‘Running'. Repeat the ‘get pods' command until the status is ‘Running'. This particular pod typically takes around 40 to 45 seconds to start in this configuration.

**Note**

The Bookshelf pods will have a name similar to bookshelf-frontend-9xx74. To review the status of a particular Bookshelf pod, you may also type:

**kubectl get pods guestbook-9xx74**

## Step 11

After a few moments, the external IP of the load balancer is listed in the EXTERNAL-IP column of the service. Type the following kubectl command to retrieve the IP address.

kubectl get services bookshelf-frontend

Note the external IP address of the service. You use it later. If the external address is not visible, repeat the "get services" command.

## Step 12

Open a new tab in your browser.

## Step 13

In the address bar, type the external IP address returned by the "get services" command.

The Bookshelf web application loads.

## Step 14

Switch to your Cloud Shell window, and type the following command to close it.

exit

Leave all other browser windows open.

5. Test Bookshelf

Generate some simple test data, including a sample cover image, in the Bookshelf application, to verify that it is working as expected.

To test Bookshelf:

Step 1

Return to the Bookshelf browser tab. You may, or may not see existing book details in the bookshelf, based on any previous labs you may have completed earlier in the course.

Click **Add Book**.

Step 2

In the ‘Add book' page:

* For **Title**, type **CPB200 Google BigQuery for Data Analysts**.
* For **Author**, type **John Smith**.
* For **Date Published**, type today's date.
* For **Description**, type **A companion study guide to the CPB200 course**.
* For **Cover Image**, upload the file, cpb200cover.jpg.

[DOWNLOAD COVER IMAGE](https://storage.googleapis.com/cloud-training/CP100/Bookshelf/cpb200cover.jpg)

**Note**

The link above can be used to retrieve the specific image for this step. Alternatively you can download a ZIP archive containing all of the sample images for this course using the button below. You will need to unzip the archive to upload the individual image required in this lab.

[DOWNLOAD ZIP](https://storage.googleapis.com/cloud-training/CP100/Bookshelf/bookcovers.zip)

* Click **Save**.

A page loads with the details you recorded for the new book, in addition to the sample cover image.

Step 3

Close the Bookshelf browser tab, but leave the Cloud Platform Console open.

6. Compare Bookshelf versions

Before you start working through the ‘clean up' section of this lab, it is useful to take a few moments to consider the major differences between the versions of Bookshelf you have experimented with so far on this course. Can you think of any others that are important to you?

|  |  |  |
| --- | --- | --- |
| **Bookshelf version** | **App Engine standard runtime** | **Container Engine** |
| Runtimes | Sandboxed version of Python 2.7 | Custom Python runtime running in Docker |
| Access to Datastore | [NDB](https://cloud.google.com/appengine/docs/python/ndb/) | [Google Cloud Client Library](https://cloud.google.com/sdk/cloud-client-libraries) |
| Access to Cloud Storage | [Cloud Storage Client Library](https://cloud.google.com/appengine/docs/python/googlecloudstorageclient/using-cloud-storage) | [Google Cloud Client Library](https://cloud.google.com/sdk/cloud-client-libraries) |
| Notable advantages | * Minimal operational effort required * Rapid development and deployment experience | * Highly flexible deployment environment * Light coupling of application and operating system layers |

## 7. Clean up

To clean up the resources used in the lab:

## Step 1

Return to the Cloud Platform Console and in the navigator pane, click **Compute > Container Engine**.

## Step 2

Click the check box to the left of the cluster name. This will activate the Delete button at the top of the page.

## Step 3

Click **Delete** to remove the Container Cluster and the associated Compute Engine instances.

## Step 4

Click **Container Registry**.

## Step 5

Click **bookshelf**.

## Step 6

Check the box in the **Name** column and then click **Delete** to remove the latest image for the Bookshelf container.

## Step 7

Review the warning, type **DELETE**, and then click **Delete** to confirm the operation.

## Step 8

Click the **Compute > Networking**.

## Step 9

Click**Firewall rules**.

## Step 10

In the **Firewall rules** section, check any firewall rules with names beginning with the prefix k8s-, and then click **Delete**.

## Step 11

Click **Load balancing**.

## Step 12

You can click the load balancer to expand the details. This will confirm that the load balancer is associated with your gke instances. To the right of the load balancer, click the **Delete** button (the trash can) and then confirm your choice.

## Step 13

Close all browser tabs.

©Google, Inc. or its affiliates. All rights reserved. Do not distribute.