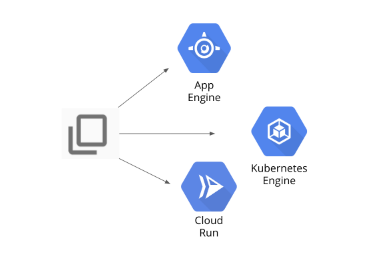
**Deploying Apps to Google Cloud**

experimentLabschedule1 hour 30 minutesuniversal\_currency\_alt1 Creditshow\_chartIntroductory

infoThis lab may incorporate AI tools to support your learning.

**Overview**

In this lab, you will deploy applications to the Google Cloud services App Engine, Kubernetes Engine, and Cloud Run.



Objectives

In this lab, you will learn how to perform the following tasks:

* + - * Download a sample app from GitHub
      * Deploy to App Engine
      * Deploy to Kubernetes Engine
      * Deploy to Cloud Run

**Set up your lab environment**

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

* + - * Sign in to Qwiklabs using an **incognito window**.
      * Note the lab's access time (for example, 1:15:00), and make sure you can finish within that time.  
        There is no pause feature. You can restart if needed, but you have to start at the beginning.
      * When ready, click **Start lab**.
      * Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
      * Click **Open Google Console**.
      * Click **Use another account** and copy/paste credentials for **this** lab into the prompts.  
        If you use other credentials, you'll receive errors or **incur charges**.
      * Accept the terms and skip the recovery resource page.

**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

**Task 1. Create a simple Python application**

You need some source code to manage. So, you will create a simple Python Flask web application. The application will be only slightly better than "hello world", but it will be good enough to test the pipeline you will build.

* + - * In the Cloud Console, click **Activate Cloud Shell** (Activate Cloud Shell icon).
      * If prompted, click **Continue**.

3.9. Enter the following command in Cloud Shell to create a folder called gcp-course:

mkdir gcp-course

Copied!

* + - 1. Change to the folder you just created:

cd gcp-course

Copied!

* + - 1. Create a folder called deploying-apps-to-gcp:

mkdir deploying-apps-to-gcp

Copied!

* + - 1. Change to the folder you just created:

cd deploying-apps-to-gcp

Copied!

* + - 1. In Cloud Shell, click **Open Editor** (Editor icon) to open the code editor. If prompted click **Open in a new window**.
      2. Select the **gcp-course > deploying-apps-to-gcp** folder in the explorer tree on the left.
      3. Click on **deploying-apps-to-gcp**.
      4. Click **New File**.
      5. Name the file main.py and press **Enter**.
      6. Paste the following code into the file you just created:

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route("/")

def main():

model = {"title": "Hello GCP."}

return render\_template('index.html', model=model)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=8080, debug=True, threaded=True)

Copied!

* + - 1. To save your changes. Press **CTRL + S**.
      2. Click on the deploying-apps-to-gcp folder.
      3. Click **New Folder**.
      4. Name the folder templates and press **Enter**.
      5. Right-click on the templates folder and create a new file called layout.html.
      6. Add the following code and save the file as you did before:

<!doctype html>

<html lang="en">

<head>

<title>{{model.title}}</title>

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">

</head>

<body>

<div class="container">

{% block content %}{% endblock %}

<footer></footer>

</div>

</body>

</html>

Copied!

* + - 1. Also in the templates folder, add another new file called index.html.
      2. Add the following code and save the file as you did before:

{% extends "layout.html" %}

{% block content %}

<div class="jumbotron">

<div class="container">

<h1>{{model.title}}</h1>

</div>

</div>

{% endblock %}

Copied!

* + - 1. In Python, application prerequisites are managed using pip. Now you will add a file that lists the requirements for this application.
      2. In the **deploying-apps-to-gcp** folder (*not the templates folder*), create a **New File** and add the following to that file and save it as requirements.txt:

Flask==2.0.3

itsdangerous==2.0.1

Jinja2==3.0.3

werkzeug==2.2.2

Copied!

**Task 2. Define a Docker build**

The first step to using Docker is to create a file called **Dockerfile**. This file defines how a Docker container is constructed. You will do that now.

* + - * In the **deploying-apps-to-gcp** folder, click **New File** and name the new file **Dockerfile**.

The file *Dockerfile* is used to define how the container is built.

* + - 1. Add the following:

FROM python:3.9

WORKDIR /app

COPY . .

RUN pip install gunicorn

RUN pip install -r requirements.txt

ENV PORT=8080

CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 main:app

Copied!

* + - 1. To test the program, type the following command to build a Docker container of the image:

docker build -t test-python .

Copied!

* + - 1. To run the Docker image, type the following command:

docker run --rm -p 8080:8080 test-python

Copied!

* + - 1. To see the program running, click  **Web Preview** (Web Preview icon) in the toolbar of Google Cloud Shell. Then, select **Preview on port 8080**.

The program should be displayed in a new browser tab.

* + - 1. In Cloud Shell, type Ctrl+C to stop the program.

**Task 3. Deploy to App Engine**

App Engine is a completely automated deployment platform. It supports many languages, including Python, Java, JavaScript, and Go. To use it, you create a configuration file and deploy your applications with a couple of simple commands. In this task, you create a file named *app.yaml* and deploy it to App Engine.

* + - * In Cloud Shell, click **Open Editor** (Cloud Shell Editor icon), then click **Open in a new window** if required.
      * Select the gcp-course/deploying-apps-to-gcp folder in the explorer tree on the left.
      * Click **New File**, name the file **app.yaml**, and then press **Enter**.
      * Paste the following into the file you just created:

runtime: python39

Copied!

* + - 1. **Save** your changes.

**Note:** There are other settings you can add to the app.yaml file, but in this case only the language runtime is required.

* + - 1. In a project, an App Engine application has to be created. This is done just once using the gcloud app create command and specifying the region where you want the app to be created. Click **Open Terminal** and type the following command. If prompted, click **Authorize**:

gcloud app create --region=us-west1

Copied!

* + - 1. Now deploy your app with the following command:

gcloud app deploy --version=one --quiet

Copied!

**Note:** This command will take a couple of minutes to complete.

* + - 1. On the Google Cloud console title bar, type **App Engine** in the Search field, then click **App Engine** in the Search Results section.
      2. In the upper-right corner of the dashboard is a link to your application, similar to this:

Example of the application link

**Note:** By default, the URL to an App Engine application is in the form of https://project-id.appspot.com.

* + - 1. Click on the link to test your program.
      2. Make a change to the program to see how easy the App Engine makes managing versions.
      3. In the code editor, expand the /deploying-apps-to-gcp folder in the navigation pane on the left. Then, click **main.py** to open it.
      4. In the **main()** function, change the title to Hello App Engine as shown below:

@app.route("/")

def main():

model = {"title" "Hello App Engine"}

return render\_template('index.html', model=model)

* + - 1. Click **File > Save** in the code editor toolbar to save your change.
      2. Now, deploy version two with the following command:

gcloud app deploy --version=two --no-promote --quiet

Copied!

**Note:** The --no-promote parameter tells App Engine to continue serving requests with the old version. This allows you to test the new version before putting it into production.

* + - 1. When the command completes, return to the App Engine dashboard. Click the link again, and version one will still be returned. It should return Hello GCP. This is because of the --no-promote parameter in the previous command.
      2. On the left, click the **Versions** tab. Notice that two versions are listed.

**Note:** You might have to click **Refresh** to see version **two**.

* + - 1. Click on the version **two** link to test it. It should return Hello App Engine.
      2. To migrate production traffic to version two, click  **Split Traffic** at the top. Change the version to **two**, and click **Save**.
      3. Give it a minute to complete. Refresh the browser tab that earlier returned Hello GCP. It should now return the new version.

Click **Check my progress** to verify the objective.

Deploy to App Engine

**Task 4. Deploy to Kubernetes Engine with Cloud Build and Artifact Registry**

Kubernetes Engine allows you to create a cluster of machines and deploy any number of applications to it. Kubernetes abstracts the details of managing machines and allows you to automate the deployment of your applications with simple CLI commands.

To deploy an application to Kubernetes, you first need to create the cluster. Then you need to add a configuration file for each application you will deploy to the cluster.

* + - * On the **Navigation menu** (Navigation menu icon), click **Kubernetes Engine**. If a message appears saying the Kubernetes API is being initialized, wait for it to complete.
      * Click **Create Cluster** then click **Switch to Standard Cluster** confirm **Switch to Standard Cluster**.
      * Click **Zonal** for **Location type** and then select the zone us-west1-c. Accept all the other variables as default then click **Create**. It will take a couple of minutes for the Kubernetes Engine cluster to be created. When the cluster is ready, a green check appears.
      * Click the three dots to the right of the cluster and then click **Connect**.
      * In the **Connect to the cluster** screen, click **Run in Cloud Shell**. This opens Cloud Shell with the connect command entered automatically.
      * Press **Enter** to connect to the cluster.
      * To test your connection, type the following command:

kubectl get nodes

Copied!

This command simply shows the machines in your cluster. If it works, you're connected.

* + - 1. In Cloud Shell, click **Open Editor** (Cloud Shell Editor icon).
      2. Expand the gcp-course/deploying-apps-to-gcp folder in the navigation pane on the left. Then, click **main.py** to open it.
      3. In the **main()** function, change the title to Hello Kubernetes Engine as shown below:

@app.route("/")

def main():

model = {"title" "Hello Kubernetes Engine"}

return render\_template('index.html', model=model)

* + - 1. **Save** your change.
      2. Add a file named kubernetes-config.yaml to the  gcp-course/deploying-apps-to-gcp folder.
      3. Paste the following code in that file to configure the application:

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: devops-deployment

labels:

app: devops

tier: frontend

spec:

replicas: 3

selector:

matchLabels:

app: devops

tier: frontend

template:

metadata:

labels:

app: devops

tier: frontend

spec:

containers:

- name: devops-demo

image: <YOUR IMAGE PATH HERE>

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: devops-deployment-lb

labels:

app: devops

tier: frontend-lb

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 8080

selector:

app: devops

tier: frontend

Copied!

**Note:** In the first section of the YAML file above, you are configuring a deployment. In this case, you are deploying 3 instances of your Python web app. Notice the image attribute. You will update this value with your image in a minute after you build it. In the second section, you are configuring a service of the type "load balancer". The load balancer will have a public IP address. Users will access your application through the load balancer.

For more information on Kubernetes deployments and services, see the links below:

* + - * [Kubernetes Deployments page](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)
      * [Kubernetes Create an External Load Balancer page](https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/)
      1. In **Cloud Shell** type the following command to create an Artifact Registry repository named devops-demo:

gcloud artifacts repositories create devops-demo \

--repository-format=docker \

--location=us-west1

Copied!

* + - 1. To configure Docker to authenticate to the Artifact Registry Docker repository, type the following command:

gcloud auth configure-docker us-west1-docker.pkg.dev

Copied!

* + - 1. To use Kubernetes Engine, you need to build a Docker image. Type the following commands to use Cloud Build to create the image and store it in Artifact Registry:

cd ~/gcp-course/deploying-apps-to-gcp

gcloud builds submit --tag us-west1-docker.pkg.dev/$DEVSHELL\_PROJECT\_ID/devops-demo/devops-image:v0.2 .

Copied!

* + - 1. When the previous command completes, the image name will be listed in the output. The image name is in the form us-west1-docker.pkg.dev/PROJECT\_ID/devops-demo/devops-image:v0.2.
      2. Highlight your image name and copy it to the clipboard. Paste that value in the kubernetes-config.yaml file, overwriting the string <YOUR IMAGE PATH HERE>.

You should see something similar to below:

spec:

containers:

- name: devops-demo

image: us-west1-docker.pkg.dev/PROJECT\_ID/devops-demo/devops-image:v0.2

ports:

* + - 1. Type the following Kubernetes command to deploy your application:

kubectl apply -f kubernetes-config.yaml

Copied!

* + - 1. In the configuration file, three replicas of the application were specified. Type the following command to see whether three instances have been created:

kubectl get pods

Copied!

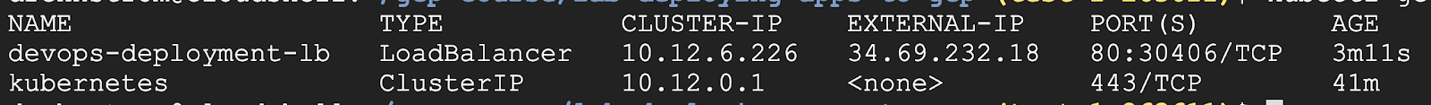
Make sure all the pods are ready. If they aren't, wait a few seconds and try again.

* + - 1. A load balancer was also added in the configuration file. Type the following command to see whether it was created:

kubectl get services

Copied!

You should see something similar to below:



If the load balancer's external IP address says "pending", wait a few seconds and try again.

* + - 1. When you have an external IP, open a browser tab and make a request to it. It should return Hello Kubernetes Engine. It might take a few seconds to be ready.

Click **Check my progress** to verify the objective.

Deploy to Kubernetes Engine

**Task 5. Deploy to Cloud Run**

Cloud Run simplifies and automates deployments to Kubernetes. When you use Cloud Run, you don't need a configuration file. You simply choose a cluster for your application. With Cloud Run, you can use a cluster managed by Google, or you can use your own Kubernetes cluster.

To use Cloud Run, your application needs to be deployed using a Docker image and it must be stateless.

* + - * Open the Cloud Shell code editor and expand the gcp-course/deploying-apps-to-gcp folder in the navigation pane on the left. Then, click **main.py** to open it.
      * In the **main()** function, change the title to Hello Cloud Run as shown below:

@app.route("/")

def main():

model = {"title" "Hello Cloud Run"}

return render\_template('index.html', model=model)

* + - 1. **Save** your change.
      2. To use Cloud Run, you need to build a Docker image. In Cloud Shell, type the following commands to use Cloud Build to create the image and store it in Artifact Registry:

cd ~/gcp-course/deploying-apps-to-gcp

gcloud builds submit --tag us-west1-docker.pkg.dev/$DEVSHELL\_PROJECT\_ID/devops-demo/cloud-run-image:v0.1 .

Copied!

* + - 1. When the build completes, on the Google Cloud console title bar, type **Cloud Run**in the Search field, then click **Cloud Run** in the Products & Pages section.
      2. Click **Create service**. This enables the Cloud Run API.
      3. Click the **Select** link in the Container image URL text box and then click **Artifact Registry**. In the resulting dialog, expand **Region-docker.pkg.dev/$DEVSHELL\_PROJECT\_ID/devops-demo** > **cloud-run-image**and select the image listed. Then click **Select**.
      4. In  **Service name**, type **hello-cloud-run** and select region us-west1.
      5. For **Authentication**, select **Allow unauthenticated invocations**.
      6. In **Container(s), Volumes, Networking, Security**, select **Default** in the **Execution environment** section.
      7. In **Revision scaling**, set the **Maximum number of instances** to **6**. Leave the rest as defaults.
      8. Finally, click **Create**.
      9. It shouldn't take long for the service to deploy. When a green check appears, click on the URL that is automatically generated for the application. It should return Hello Cloud Run.

Click **Check my progress** to verify the objective.

Deploy to Cloud Run

**Congratulations!**

In this lab, you deployed applications to the Google Cloud services App Engine, Kubernetes Engine, and Cloud Run.

**End your lab**

When you have completed your lab, click **End Lab**. Google Cloud Skills Boost 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.

Copyright 2025 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.