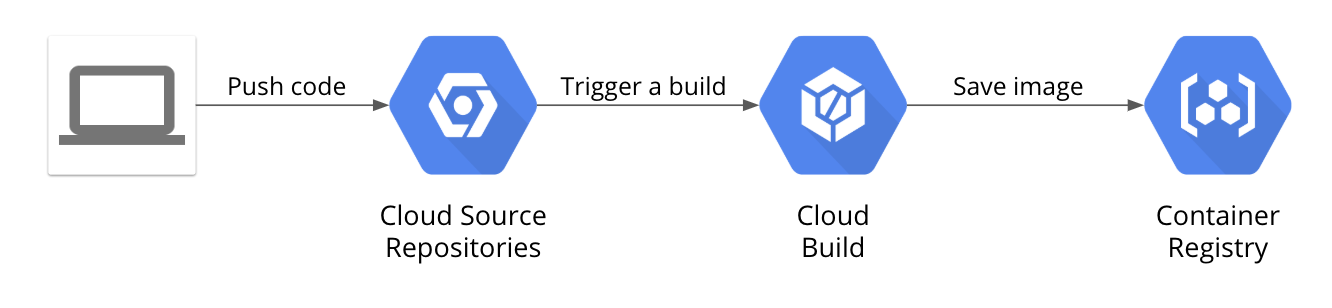
Building a DevOps Pipeline

2 hoursFree

Rate Lab

**Overview**

In this lab, you will build a continuous integration pipeline using Cloud Source Repositories, Cloud Build, build triggers, and Container Registry.



**Objectives**

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

* Create a Git repository
* Create a simple Python application
* Test Your web application in Cloud Shell
* Define a Docker build
* Manage Docker images with Cloud Build and Container Registry
* Automate builds with triggers
* Test your build changes

**Set up your lab environment**

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

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, img/time.png and make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

1. When ready, click img/start_lab.png.
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

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

**Task 1: Create a Git Repository**

First, you will create a Git repository using the Cloud Source Repositories service in Google Cloud. This Git repository will be used to store your source code. Eventually, you will create a build trigger that starts a continuous integration pipeline when code is pushed to it.

1. In the Cloud Console, on the **Navigation menu**, click **Source Repositories**. A new tab will open.
2. Click **Add repository**.
3. Select **Create new repository** and click **Continue**.
4. Name the repository **devops-repo**
5. Select your current project ID from the list.
6. Click **Create**.
7. Return to the Cloud Console, and click **Activate Cloud Shell** (Cloud Shell).
8. If prompted, click **Continue**.
9. Enter the following command in Cloud Shell to create a folder called gcp-course:

mkdir gcp-course

1. Change to the folder you just created:

cd gcp-course

1. Now clone the empty repository you just created:

gcloud source repos clone devops-repo

You will see a warning that you have cloned an empty repository. That is expected at this point.

1. The previous command created an empty folder called devops-repo. Change to that folder:

cd devops-repo

Click *Check my progress* to verify the objective.

Create a git repository.

Check my progress

**Task 2: 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.

1. In Cloud Shell, click **Launch Editor** (cloud-shell-editor.png) to open the code editor.
2. Select the **devops-repo** folder in the explorer tree on the left.
3. On the **File** menu, click **New File**, and name the file **main.py**
4. Paste the following into the file you just created:

from flask import Flask, render\_template, request

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

@app.route("/")

def main():

model = {"title": "Hello DevOps Fans."}

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

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

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

1. Save your changes.
2. Right-click on the devops-repo folder and add a new folder called templates.
3. In that folder, add a new file called layout.html.
4. 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>

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 %}

1. In Python, application prerequisites are managed using pip. Now you will add a file that lists the requirements for this application. In the **devops-repo** folder (*not the templates folder*), create a file called requirements.txt.
2. Add the following to that file and save it:

Flask==1.1.1

1. You have some files now, so save them to the repository. First, you need to add all the files you created to your local Git repo. In Cloud Shell, enter the following code:

cd ~/gcp-course/devops-repo

git add --all

1. To commit changes to the repository, you have to identify yourself. Enter the following commands, but with your information (*you can just use your Gmail address or any other email address*):

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

1. Now, commit the changes locally:

git commit -a -m "Initial Commit"

1. You committed the changes locally, but have not updated the Git repository you created in Cloud Source Repositories. Enter the following command to push your changes to the cloud:

git push origin master

1. Refresh the **Source Repositories** web page. You should see the files you just created.

**Task 3: Test Your Web Application in Cloud Shell**

You need to make sure the code works. It can be tested using Cloud Shell.

1. Back in Cloud Shell, make sure you are in your application's root folder, and then install the Flask framework using pip:

cd ~/gcp-course/devops-repo

sudo pip3 install -r requirements.txt

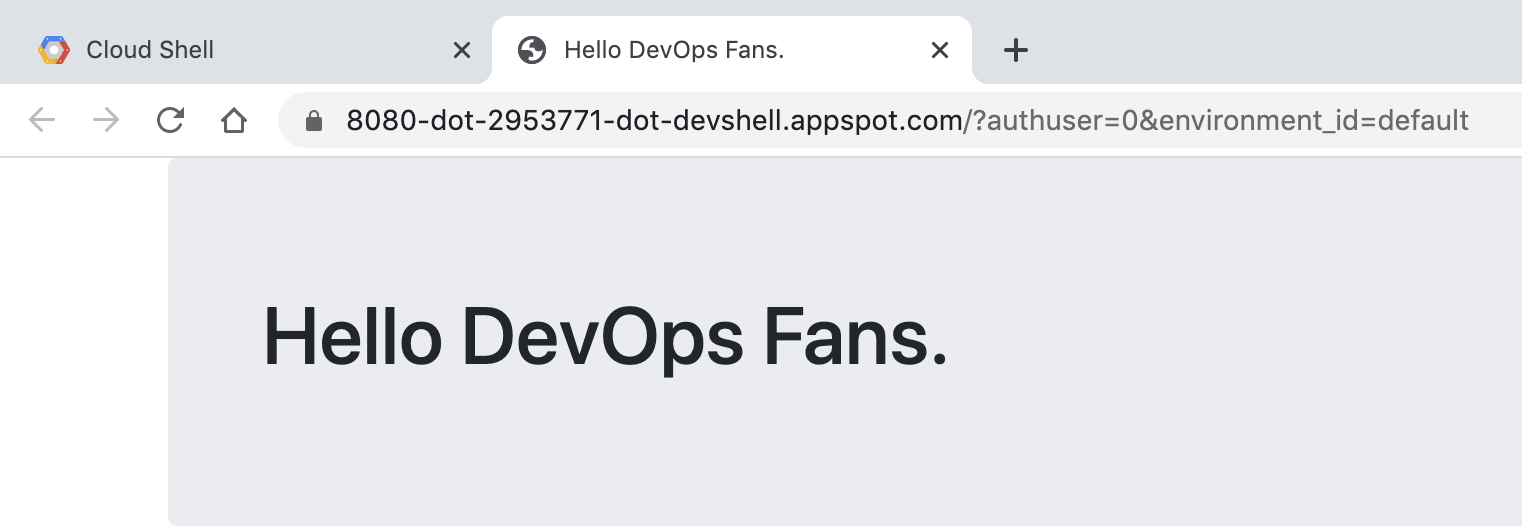
1. To run the program, type:

python3 main.py

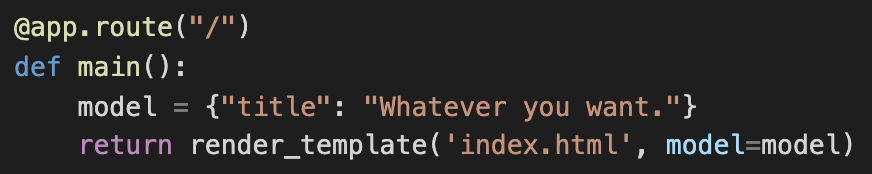
**Note:**The program is configured to run on port 8080. You can see this if you look at the main.py file. At the bottom, when the app runs, the port is set.

1. To see the program running, click **Web Preview** (web-preview.png.png) in the toolbar of Cloud Shell. Then, click **Preview on port 8080**.

The program should be displayed in a new browser tab.



1. To stop the program, return to the Cloud Console and press Ctrl+C in Cloud Shell.
2. In the code editor, expand the **gcp-course/devops-repo** folder in the explorer pane on the left. Then, click **main.py** to open it.
3. In the **main()** function, change the title to something else (*whatever you want*), as shown below.



1. On the Code Editor toolbar, on the **File** menu, click **Save** to save your change.
2. In the Cloud Shell window at the bottom, commit your changes using the following commands:

cd ~/gcp-course/devops-repo

git commit -a -m "Second Commit"

1. Push your changes to the cloud using the following command:

git push origin master

1. Return to the **Source Repositories** page and refresh the repository to verify that your changes were uploaded.

**Task 4: 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.

1. In the Cloud Shell Code Editor, expand the **gcp-course/devops-repo** folder. With the **devops-repo** folder selected, On the **File** menu, click **New File** and name the new file **Dockerfile**

The file *Dockerfile* is used to define how the container is built. 2. At the top of the file, enter the following:

FROM python:3.7

This is the base image. You could choose many base images. In this case, you are using one with Python already installed on it.

1. Enter the following:

WORKDIR /app

COPY . .

These lines copy the source code from the current folder into the /app folder in the container image.

4.Enter the following:

RUN pip install gunicorn

RUN pip install -r requirements.txt

This uses pip to install the requirements of the Python application into the container. Gunicorn is a Python web server that will be used to run the web app.

1. Enter the following:

ENV PORT=80

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

The environment variable sets the port that the application will run on (in this case, 80). The last line runs the web app using the gunicorn web server.

1. Verify that the completed file looks as follows and save it:

FROM python:3.7

WORKDIR /app

COPY . .

RUN pip install gunicorn

RUN pip install -r requirements.txt

ENV PORT=80

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

**Task 5: Manage Docker Images with Cloud Build and Container Registry**

The Docker image has to be built and then stored somewhere. You will use **Cloud Build** and **Container Registry**.

1. Return to Cloud Shell. Make sure you are in the right folder:

cd ~/gcp-course/devops-repo

1. The Cloud Shell environment variable DEVSHELL\_PROJECT\_ID automatically has your current project ID stored. The project ID is required to store images in Container Registry. Enter the following command to view your project ID:

echo $DEVSHELL\_PROJECT\_ID

1. Enter the following command to use Cloud Build to build your image:

gcloud builds submit --tag gcr.io/$DEVSHELL\_PROJECT\_ID/devops-image:v0.1 .

*Notice the environment variable in the command*. The image will be stored in Container Registry.

1. If asked to enable Cloud Build in your project, type **Yes**. Wait for the build to complete successfully.

If you receive the error "INVALID\_ARGUMENT: unable to resolve source," wait a few minutes and try again.

**Note**: In Container Registry, the image name always begins with **gcr.io/**, followed by the project ID of the project you are working in, followed by the image name and version.

The period at the end of the command represents the path to the Dockerfile: in this case, the current directory.

1. Return the the **Cloud Console** and on the Navigation menu ( menu.png), click **Container Registry**. Your image should be on the list.
2. Now navigate to the **Cloud Build** service, and your build should be listed in the history.

You will now try running this image from a Compute Engine virtual machine.

1. Navigate to the **Compute Engine** service.
2. Click **Create** to create a VM.
3. On the **Create an instance** page, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** |
| **Container** | Deploy a container image to this VM instance |
| **Container image** | gcr.io/<your-project-id-here>/devops-image:v0.1 (change the project ID where indicated) |
| **Firewall** | Allow HTTP traffic |

1. Click **Create**.
2. Once the VM starts, create a browser tab and make a request to this new VM's external IP address. The program should work as before.

You might have to wait a minute or so after the VM is created for the Docker container to start.

1. You will now save your changes to your Git repository. In Cloud Shell, enter the following to make sure you are in the right folder and add your new Dockerfile to Git:

cd ~/gcp-course/devops-repo

git add --all

1. Commit your changes locally:

git commit -am "Added Docker Support"

1. Push your changes to Cloud Source Repositories:

git push origin master

1. Return to Cloud Source Repositories and verify that your changes were added to source control.

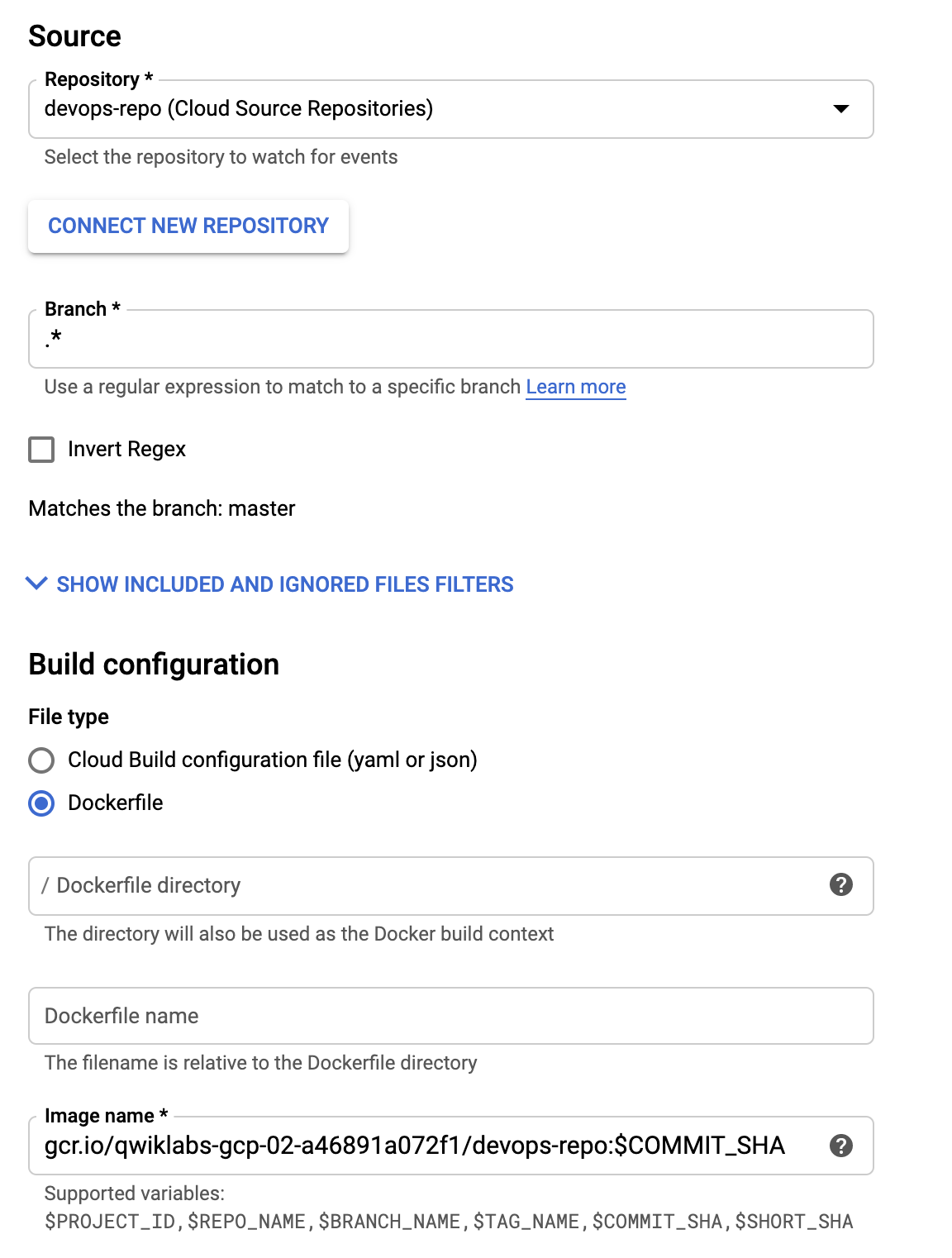
Click *Check my progress* to verify the objective.

Manage Docker images with Cloud Build and Container Registry.

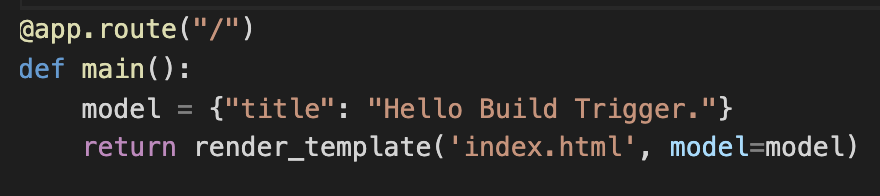
Check my progress

**Task 6: Automate Builds with Triggers**

1. On the Navigation menu (menu.png), click **Container Registry**. At this point, you should have a folder named **devops-image** with at least one container in it.
2. On the Navigation menu, click **Cloud Build**. The **Build history** page should open, and one or more builds should be in your history.
3. Click the **Triggers** link on the left.
4. Click **Create trigger**.
5. Name the trigger devops-trigger
6. Select your **devops-repo** Git repository.
7. Select **.\*(any branch)** for the branch.
8. Choose **Dockerfile** for **Build Configuration** and select the dafault image.



1. Accept the rest of the defaults, and click **Create trigger**.
2. To test the trigger, click **Run trigger**.
3. Click the **History** link and you should see a build running. Wait for the build to finish, and then click the link to it to see its details.
4. Scroll down and look at the logs. The output of the build here is what you would have seen if you were running it on your machine.
5. Return to the Container Registry service. You should see a new folder, **devops-repo**, with a new image in it.
6. Return to the **Cloud Shell Code Editor**. Find the file main.py in the gcp-course/devops-repo folder.
7. In the main() function, Change the title property to "Hello Build Trigger." as shown below:



1. Commit the change with the following command:

cd ~/gcp-course/devops-repo

git commit -a -m "Testing Build Trigger"

1. Enter the following to push your changes to Cloud Source Repositories:

git push origin master

1. Return to the Cloud Console and the **Cloud Build** service. You should see another build running.

Click *Check my progress* to verify the objective.

Automate Build with Trigger.

Check my progress

**Task 7: Test Your Build Changes**

1. When the build completes, click on it to see its details. Under **Execution Details**, copy the image link.
2. Go to the **Compute Engine** service. As you did earlier, create a new virtual machine to test this image. Select the box to deploy a container image to the VM, and paste the image you just copied.
3. Select **Allow HTTP traffic**.
4. When the machine is created, test your change by making a request to the VM's external IP address in your browser. Your new message should be displayed.

You might have to wait a few minutes after the VM is created for the Docker container to start.

Click *Check my progress* to verify the objective.

Test your Build Changes.

Check my progress