What GCP provides

IAAS. Infrastructure as a Service(pay for what they allocate)

PAAS Platform as a service (pay for what you use.)

**Google network divided :**

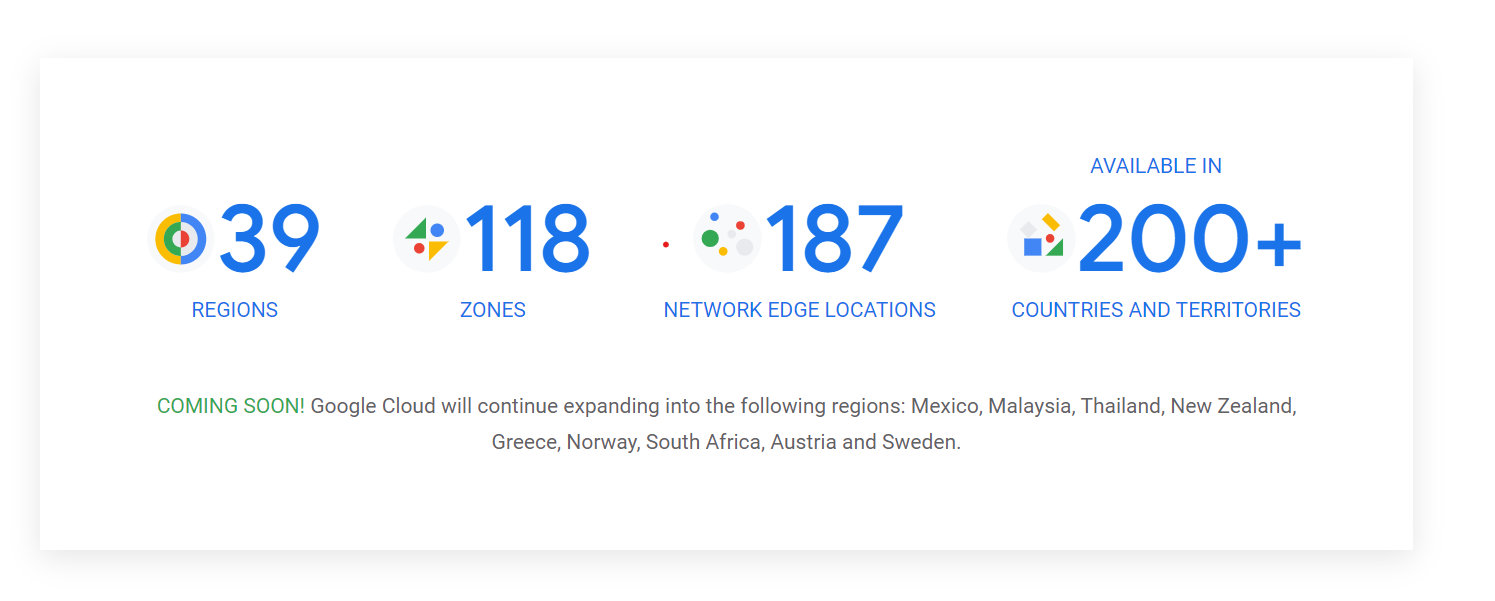
Locations

Regions

Zones: not necessarily geographically separated.

Ex: Europe-west2-a, Europe-west2-b, Europe-west2-c

Multi region resources can be used for fail over

A map of the world

Description automatically generated

**Edge location:**

End points where the fiber optics ends, and internet provider or telecom providers start.

Transport Layer Security (TLS): This protocol is used by Google Cloud to encrypt requests before transmission and protect raw data.

**Organizations**

The organization resource is the hierarchical super node of projects.

Projects should be inside organizations.

Project ids are globally unique.

While creating the project we can select different billing account.

**Cloud shell**

its free VM

Project name and id better set same

1)Folder

2)Projects

3) Resources

**Policies**

An Identity and Access Management (IAM) policy, which specifies access controls for Google Cloud resources.

A Policy is a collection of bindings. A binding binds one or more members, or principals, to a single role. Principals can be user accounts, service accounts, Google groups, and domains (such as G Suite). A role is a named list of permissions; each role can be an IAM predefined role or a user-created custom role.

A screenshot of a computer program

Description automatically generated

Policy inheritance

IAM we can view principal and roles and from where its inherited from.

Granting access in done in IAM screen

Policies can be applied to

1)Organization

2)Project

3)Resource

**IAM ROLES**

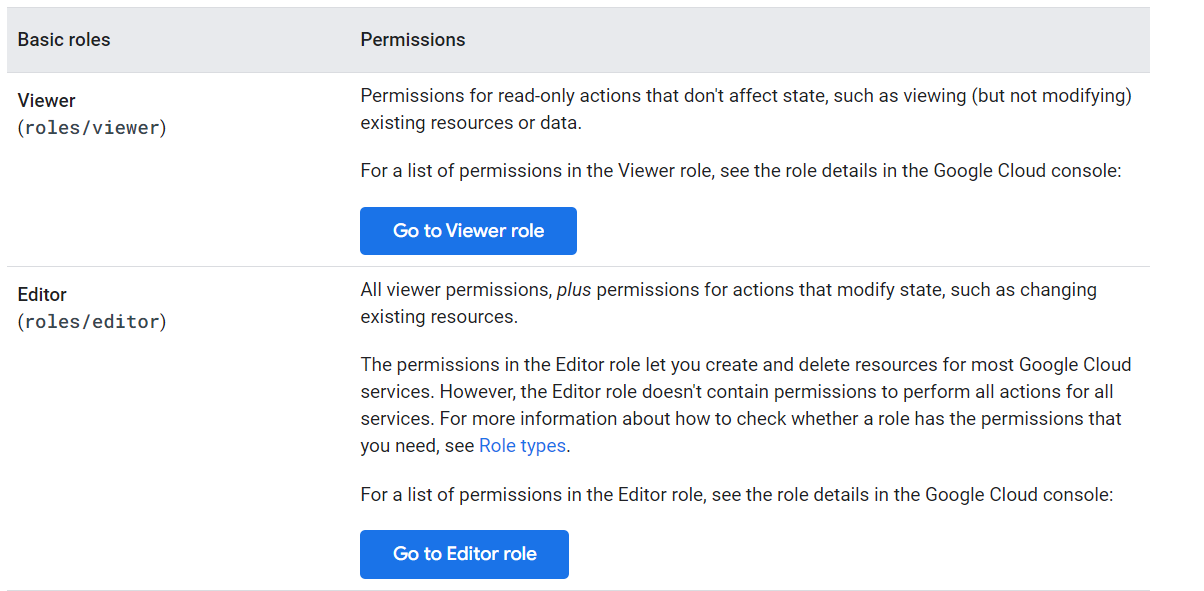
Basic Roles

Predefined Roles

Custom Roles

**Basic Roles:**

Basic roles are highly permissive roles that existed prior to the introduction of IAM. You can use basic roles to grant principals broad access to Google Cloud resources.



A screenshot of a computer

Description automatically generated

**Predefined Roles:**

Predefined roles give granular access to specific Google Cloud resources. These roles are created and maintained by Google. Google automatically updates their permissions as necessary, such as when Google Cloud adds new features or services.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Service accounts**

A Google Cloud Platform (GCP) service account is a special type of Google account that represents a non-human user. It's similar to a user account, but it represents a GCP service instead of an individual user.

service accounts can contain multiple roles.

Example allow access to cloud storage to VM.

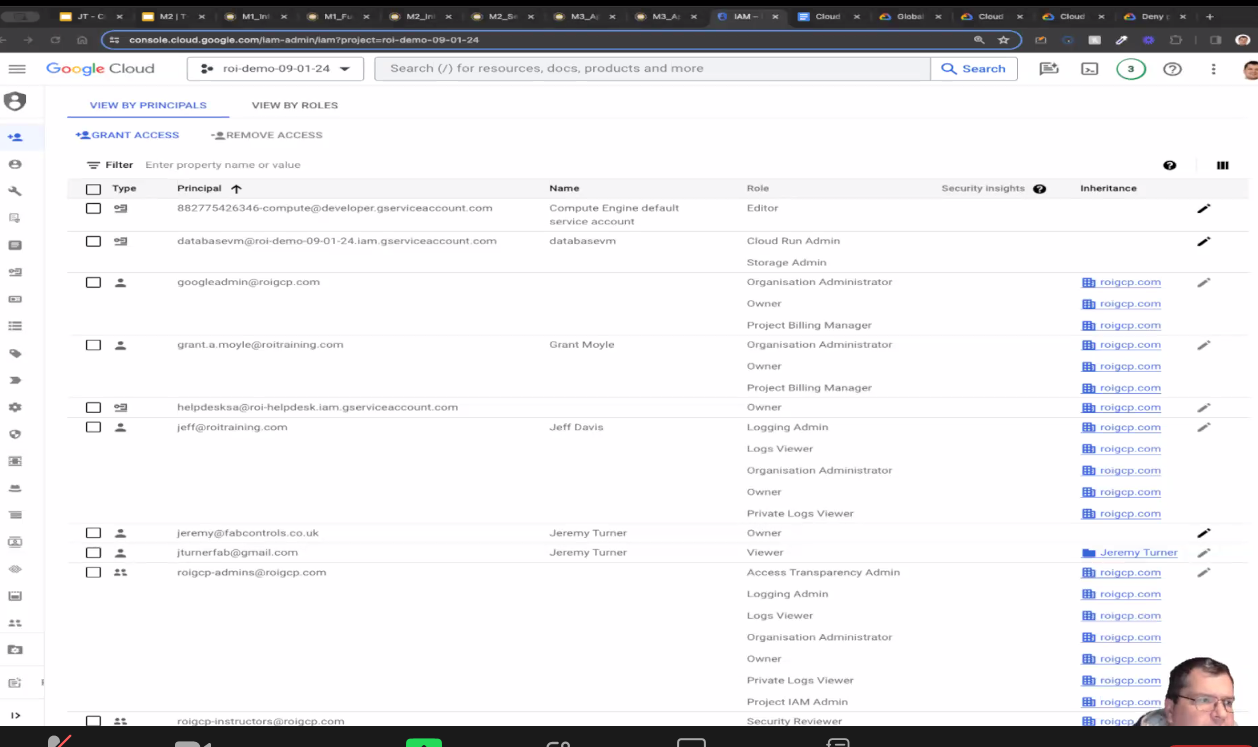
Roles are selected.

ex : cloud storage administrator

Roles contains a lot of permissions.

Service accounts are listed in IAM.

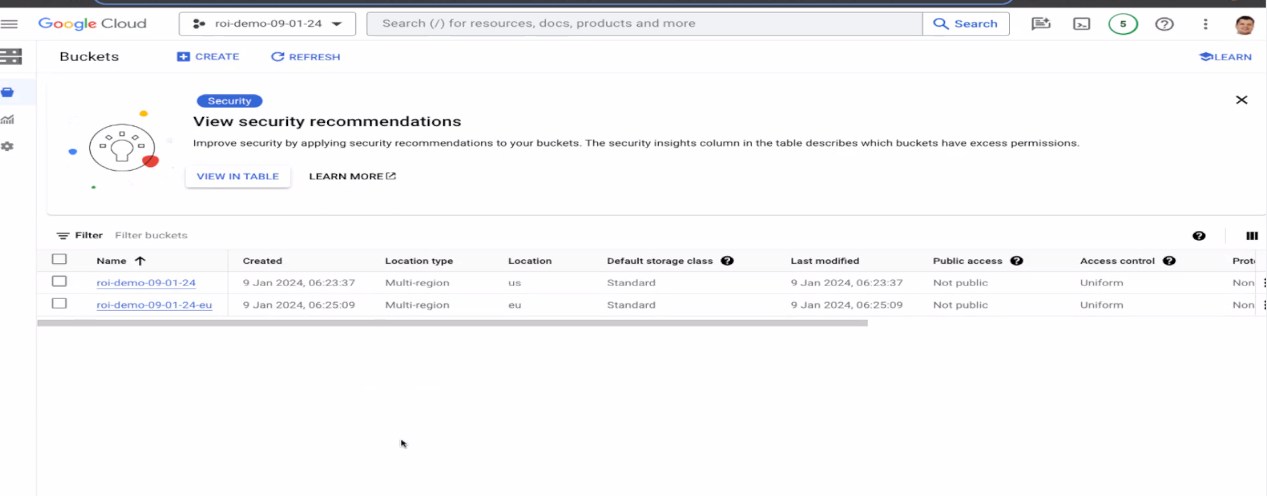
One VM can contain multiple service accounts.



Global view

We can see resources created across multiple region

I AM is a global service.



**Global resources**

Global resources are accessible by any resource in any zone within the same project. When you create a global resource, you don't need to provide a scope specification. Global resources include:

**Addresses**

The Addresses collection contains any global static external IP addresses that you have reserved for your project. Global static external IP addresses are a global resource and are used for global load balancers.

**Images**

Images are used by any instance or disk resource in the same project as the image. Google provides preconfigured images that you can use to boot your instance. You can customize one of these images, or you can build your own image. Optionally, you can share images across projects.

**Snapshots**

Persistent disk snapshots are available to all disks within the same project as the snapshot. Optionally, you can share snapshots across projects.

**Global instance templates**

An instance template is a way to save the configuration of a virtual machine (VM).

A global instance template can be used to create VM instances, managed instance groups, and reservations. You can specify zonal resources in a global instance template, which restricts the use of that template to the location of the specified zonal resource.

**Cloud Interconnects**

A Cloud Interconnect is a highly available connection from your on-premises network to Google's network. This connection is a global resource. However, interconnect attachments, which run inside of this connection, are regional resources.

**Cloud Interconnect locations**

A Cloud Interconnect location is a physical connection point for Cloud Interconnect near your network. There is one Cloud Interconnect location for every available colocation facility and edge availability domain. Cloud Interconnect locations are read-only, global resources.

**VPC network**

A VPC network is a global resource, but individual subnets are regional resources.

**Firewalls**

Firewalls apply to a single VPC network and are considered a global resource because packets can reach them from other networks.

**Routes**

Routes let you create complex networking scenarios. You can manage how traffic is routed for a specific IP range. Routes are similar to how a router directs traffic within a local area network. Routes apply to VPC networks within a Google Cloud project and are considered global resources.

**Global operations**

An operation is a per-zone resource, a per-region resource, or a global resource. If you are performing an operation on a global resource, the operation is considered a global operation. For example, inserting an image is considered a global operation because images are a global resource.

**Regional resources**

Regional resources are accessible by any resources within the same region. For example, if you reserve a static external IP address in a specific region, that static external IP address can only be assigned to instances within that region. Each region also has one or more zones. For a list of available regions and zones, see Regions and zones.

Regional resources include:

**Addresses**

The Addresses collection contains any regional static external IP addresses that you have reserved for your project. Static external IP addresses are a regional resource that are used by instances that are in the same region as the address, by regional forwarding rules for regional load balancers, and for protocol forwarding.

**Placement policies**

A placement policy controls how closely to place VMs in relation to one another. This can help reduce the impact of host system failures or network latency.

**Regional instance templates**

A regional instance template can be used to create VM instances, managed instance groups, and reservations. You can specify zonal resources in an instance template, which restricts the use of that template to the location of the specified zonal resource.

**Regional managed instance groups**

Regional managed instance groups are collections of identical instances that span multiple zones. Regional managed instance groups let you spread app load across multiple zones, rather than confining your app to a single zone or having to manage multiple instance groups across different zones.

**Regional persistent disks**

Regional persistent disks provide durable storage and replication of data between two zones within the same region. In a failover situation, you can force-attach a regional persistent disk to another instance within the same region. You cannot force attach a zonal persistent disk to an instance. Optionally, you can share disk resources across projects, which lets other projects make images and snapshots from these disks but doesn't let instances in other projects attach the disks.

**Subnets**

Subnets regionally segment the network IP space into prefixes (subnets) and control which prefix an instance's internal IP address is allocated from.

A subnet, or subnetwork, is a network inside a network. Subnets make networks more efficient. Through subnetting, network traffic can travel a shorter distance without passing through unnecessary routers to reach its destination.

**Zonal resources**

Resources that are hosted in a zone are called per-zone resources. Zone-specific resources, or per-zone resources, are unique to that zone and are only usable by other resources in the same zone. For example, an instance is a per-zone resource. When you create an instance, you must provide the zone where the instance is located. The instance can access other resources within the same zone, and can access global resources, but it can't access other per-zone resources in a different zone, such as a disk resource.

For a list of available zones, see Regions and zones.

Note: One exception is that instances in one zone can communicate with instances in another zone if both instances belong to the same VPC network.

Per-zone resources include:

**Instances**

A virtual machine (VM) instance is located within a zone and can access global resources or resources within the same zone.

**Persistent disks**

Persistent disks are accessed by other instances within the same zone. You can attach a disk only to instances in the same zone as the disk. You can't attach a disk to an instance in another zone. Optionally, you can share disk resources across projects, which lets other projects make images and snapshots from these disks but doesn't let instances in other projects attach the disks.

**Machine types**

Machine types are per-zone resources. Instances and disks can only use machine types that are in the same zone.

**Zonal managed instance groups**

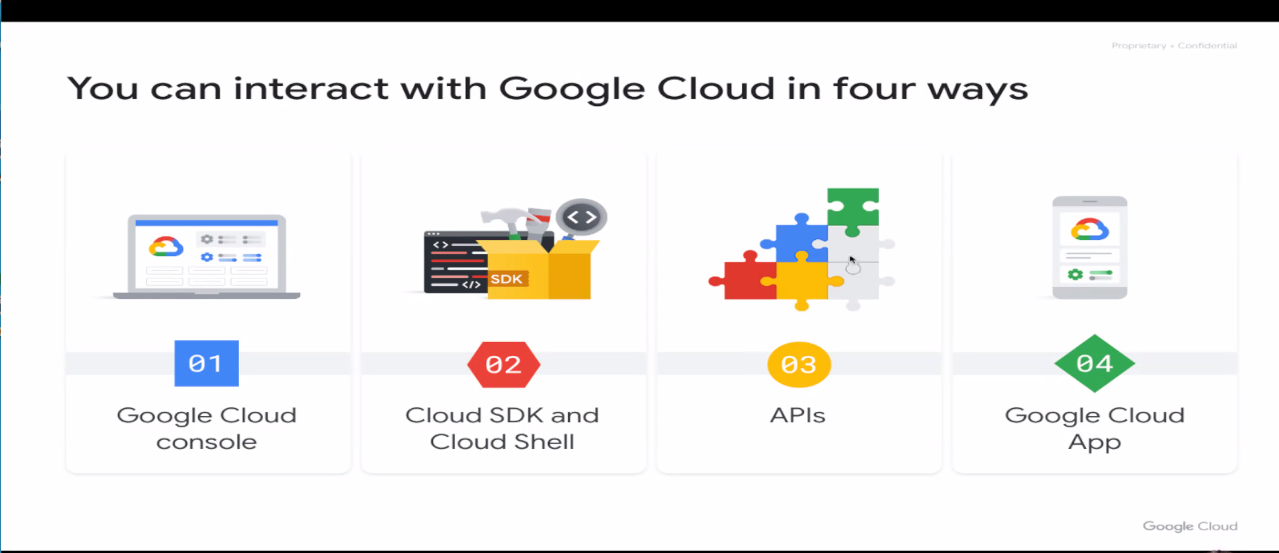
A zonal managed instance group uses an instance template to create a group of identical instances within a single zone. You manage VM instances in a managed instance group as a single entity, rather than managing individual instances.

**Cloud TPUs (Tensor Processing Units)**

TPUs are zonal resources. For information about the zones in which TPUs are available, see Availability.

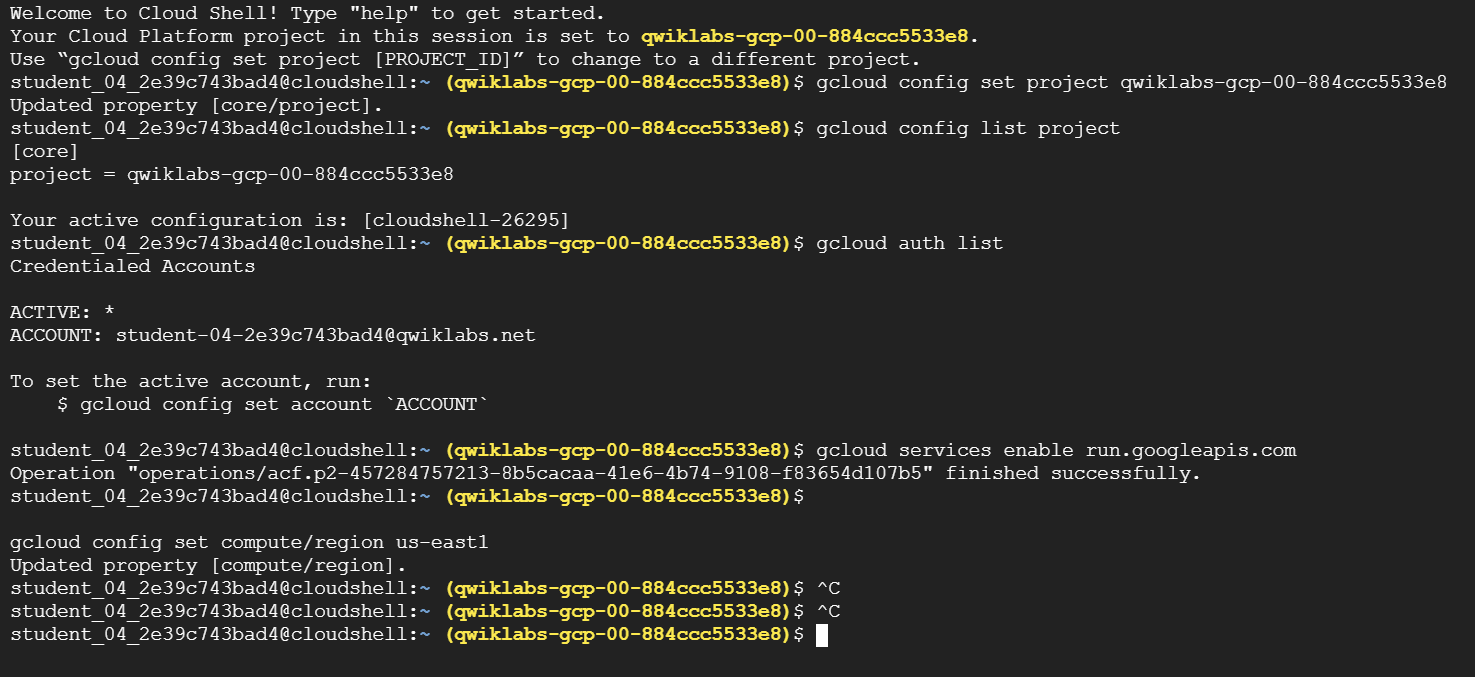
**Per-zone operations**

An operation is a per-zone resource, a per-region resource, or a global resource. If you are performing an operation on a zone-specific resource, the operation is considered a per-zone operation. For example, inserting an instance is considered a per-zone operation because the operation is being performed on a zone-specific resource, an instance.



Cloud Console -> a web-based interface that allows users to manage their Google Cloud projects and resources.

Cloud SDK and Cloud Shell -> Google Cloud CLI lets you manage resources and services from the command line.



Hello Cloud Run

25 minutes1 Credit

**Overview**



[Cloud Run](https://cloud.google.com/run) is a managed compute platform that enables you to run stateless containers that are invocable via HTTP requests. Cloud Run is serverless: it abstracts away all infrastructure management, so you can focus on what matters most — building great applications.

[Cloud Run](https://cloud.google.com/run) is built from [Knative](https://cloud.google.com/knative/" \t "_blank), letting you choose to run your containers either fully managed with Cloud Run, or in your [Google Kubernetes Engine](https://cloud.google.com/kubernetes) cluster with Cloud Run on GKE.

The goal of this lab is for you to build a simple containerized application image and deploy it to Cloud Run.

Objectives

In this lab, you learn to:

* Enable the Cloud Run API.
* Create a simple Node.js application that can be deployed as a serverless, stateless container.
* Containerize your application and upload to Container Registry (now called "Artifact Registry.")
* Deploy a containerized application on Cloud Run.
* Delete unneeded images to avoid incurring extra storage charges.

**Setup and requirements**

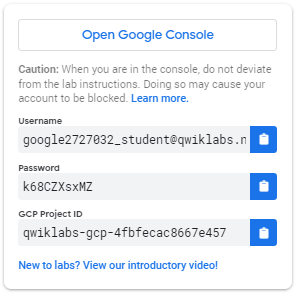
For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. 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.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. 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**.
7. 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.

**How to start your lab and sign in to the Console**

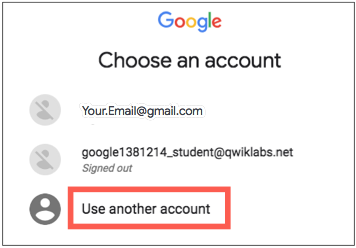
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

**Note:**Open the tabs in separate windows, side-by-side.

1. On the Choose an account page, click **Use Another Account**. The Sign in page opens.



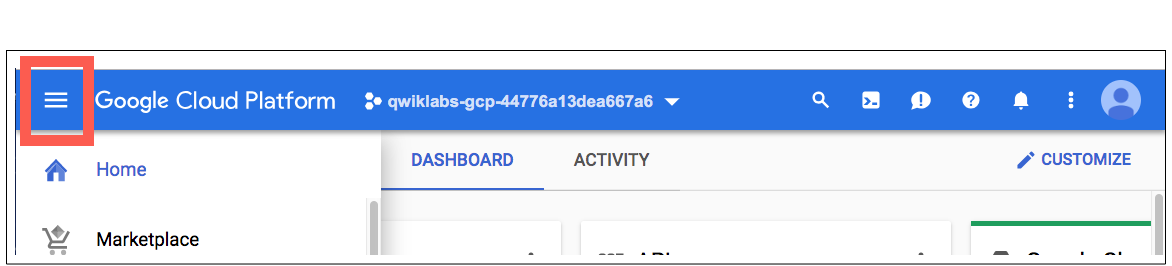
1. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

**Note:**You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

1. Click through the subsequent pages:

* Accept the terms and conditions.
* Do not add recovery options or two-factor authentication (because this is a temporary account).
* Do not sign up for free trials.

After a few moments, the Cloud console opens in this tab.

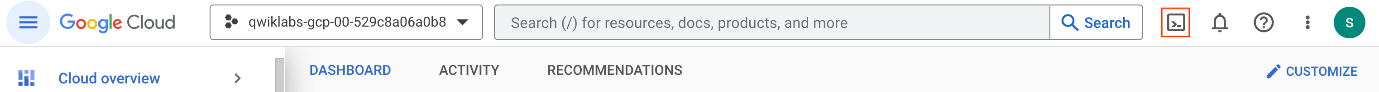
**Note:**You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

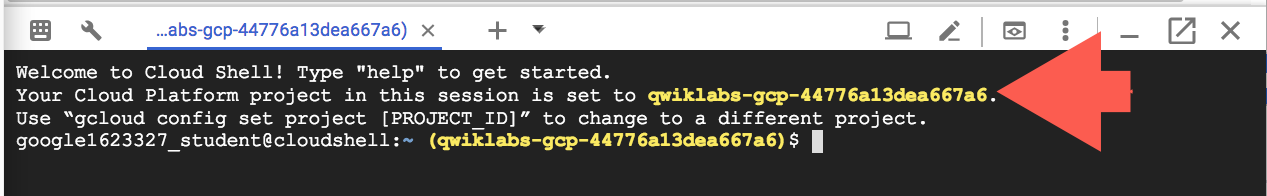
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

Reference

Basic Linux Commands

Below you will find a reference list of a few very basic Linux commands which may be included in the instructions or code blocks for this lab.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command -->** | **Action** | **.** | **Command -->** | **Action** |
| **mkdir** (*make directory*) | create a new folder | . | **cd** (*change directory*) | change location to another folder |
| **ls** (*list* ) | list files and folders in the directory | . | **cat** (*concatenate*) | read contents of a file without using an editor |
| **apt-get update** | update package manager library | . | **ping** | signal to test reachability of a host |
| **mv** (*move* ) | moves a file | . | **cp** (*copy*) | makes a file copy |
| **pwd** (*present working directory* ) | returns your current location | . | **sudo** (*super user do*) | gives higher administration privileges |

**Task 1. Enable the Cloud Run API and configure your Shell environment**

1. From Cloud Shell, enable the **Cloud Run API** :

gcloud services enable run.googleapis.com

1. If you are asked to authorize the use of your credentials, do so. You should then see a successful message similar to this one:

Operation "operations/acf.cc11852d-40af-47ad-9d59-477a12847c9e" finished successfully.

**Note:**You can also enable the API using the **APIs & Services**section of the console.

1. Set the compute region:

The command gcloud config set compute/region <region> can be used to specify a default region for the current account

gcloud config set compute/region "REGION"

1. Create a LOCATION environment variable:

LOCATION="Region"

**Task 2. Write the sample application**

In this task, you will build a simple express-based NodeJS application which responds to HTTP requests.

1. In Cloud Shell create a new directory named helloworld, then move your view into that directory:

mkdir helloworld && cd helloworld

1. Next you'll be creating and editing files. To edit files, use nano or the Cloud Shell Code Editor by clicking on the **Open Editor** button in Cloud Shell.
2. Create a package.json file, then add the following content to it:

nano package.json

{

"name": "helloworld",

"description": "Simple hello world sample in Node",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"start": "node index.js"

},

"author": "Google LLC",

"license": "Apache-2.0",

"dependencies": {

"express": "^4.17.1"

}

}

Most importantly, the file above contains a start script command and a dependency on the Express web application framework.

1. Press **CTRL+X**, then **Y**, then **Enter** to save the package.json file.
2. Next, in the same directory, create a index.js file, and copy the following lines into it:

nano index.js

const express = require('express');

const app = express();

const port = process.env.PORT || 8080;

app.get('/', (req, res) => {

const name = process.env.NAME || 'World';

res.send(`Hello ${name}!`);

});

app.listen(port, () => {

console.log(`helloworld: listening on port ${port}`);

});

This code creates a basic web server that listens on the port defined by the PORT environment variable. Your app is now finished and ready to be containerized and uploaded to Container Registry.

1. Press **CTRL+X**, then **Y**, then **Enter** to save the index.js file.

**Note:**You can use many other languages to get started with Cloud Run. You can find instructions for Go, Python, Java, PHP, Ruby, Shell scripts, and others from the [Quickstarts guide](https://cloud.google.com/run/docs/quickstarts/build-and-deploy" \t "_blank).

**Task 3. Containerize your app and upload it to Artifact Registry**

1. To containerize the sample app, create a new file named Dockerfile in the same directory as the source files, and add the following content:

nano Dockerfile

# Use the official lightweight Node.js 12 image.

# https://hub.docker.com/\_/node

FROM node:12-slim

# Create and change to the app directory.

WORKDIR /usr/src/app

# Copy application dependency manifests to the container image.

# A wildcard is used to ensure copying both package.json AND package-lock.json (when available).

# Copying this first prevents re-running npm install on every code change.

COPY package\*.json ./

# Install production dependencies.

# If you add a package-lock.json, speed your build by switching to 'npm ci'.

# RUN npm ci --only=production

RUN npm install --only=production

# Copy local code to the container image.

COPY . ./

# Run the web service on container startup.

CMD [ "npm", "start" ]

1. Press **CTRL+X**, then **Y**, then **Enter** to save the Dockerfile file.
2. Now, build your container image using Cloud Build by running the following command from the directory containing the Dockerfile. (Note the $GOOGLE\_CLOUD\_PROJECT environmental variable in the command, which contains your lab's Project ID):

gcloud builds submit --tag gcr.io/$GOOGLE\_CLOUD\_PROJECT/helloworld

Cloud Build is a service that executes your builds on GCP. It executes a series of build steps, where each build step is run in a Docker container to produce your application container (or other artifacts) and push it to Cloud Registry, all in one command.

Once pushed to the registry, you will see a SUCCESS message containing the image name (gcr.io/[PROJECT-ID]/helloworld). The image is stored in Artifact Registry and can be re-used if desired.

1. List all the container images associated with your current project using this command:

gcloud container images list

1. Register gcloud as the credential helper for all Google-supported Docker registries:

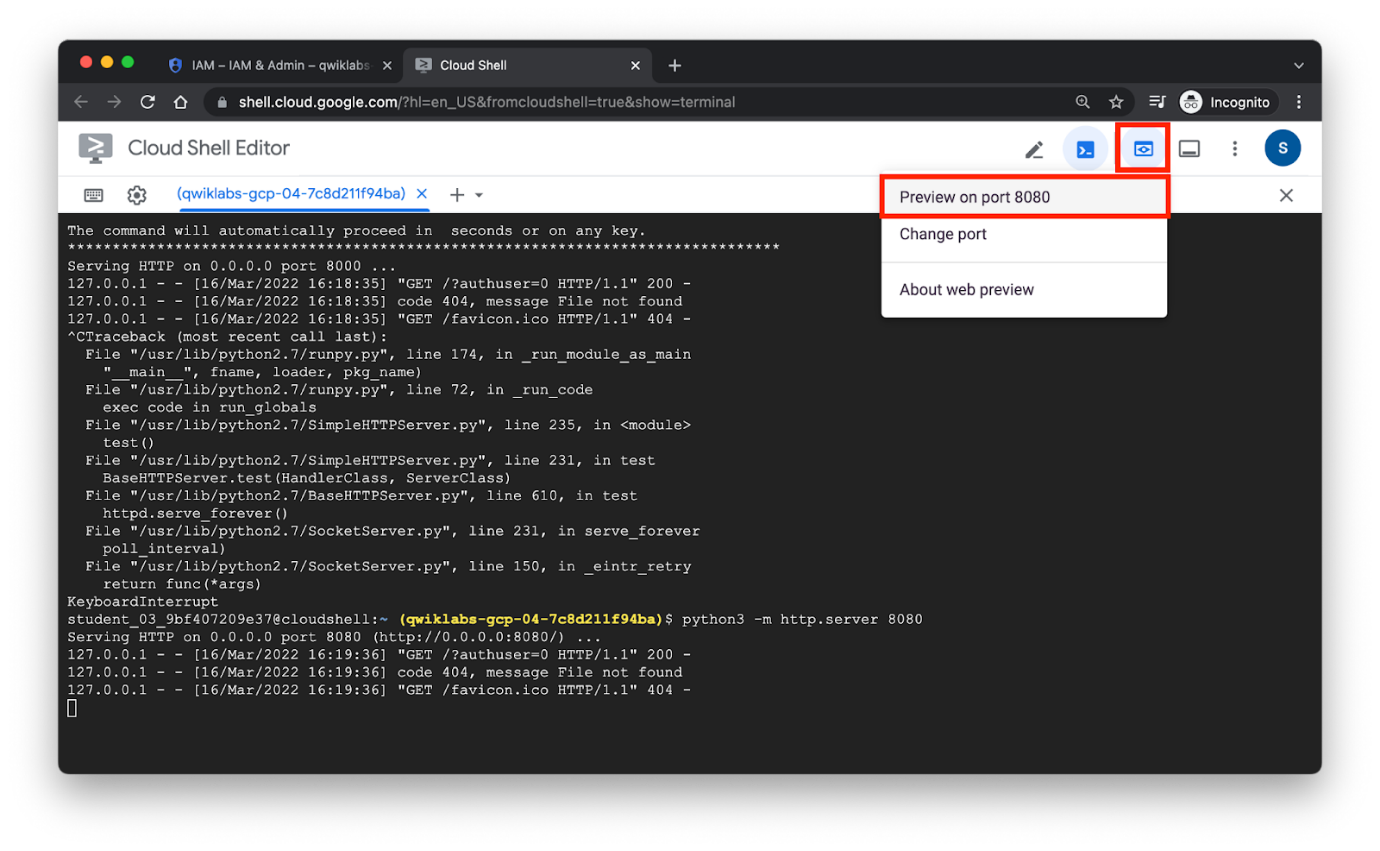
gcloud auth configure-docker

1. To run and test the application locally from Cloud Shell, start it using this standard docker command:

docker run -d -p 8080:8080 gcr.io/$GOOGLE\_CLOUD\_PROJECT/helloworld

1. In the Cloud Shell window, click on **Web preview** and select **Preview on port 8080**.

This should open a browser window showing the "Hello World!" message. You could also simply use curl localhost:8080.



**Task 4. Deploy to Cloud Run**

1. Deploying your containerized application to Cloud Run is done using the following command adding your Project-ID:

gcloud run deploy --image gcr.io/$GOOGLE\_CLOUD\_PROJECT/helloworld --allow-unauthenticated --region=$LOCATION

The allow-unauthenticated flag in the command above makes your service publicly accessible.

1. When prompted confirm the service name by pressing **Enter**.

Wait a few moments until the deployment is complete.

On success, the command line displays the service URL:

Service [helloworld] revision [helloworld-00001-xit] has been deployed

and is serving 100 percent of traffic.

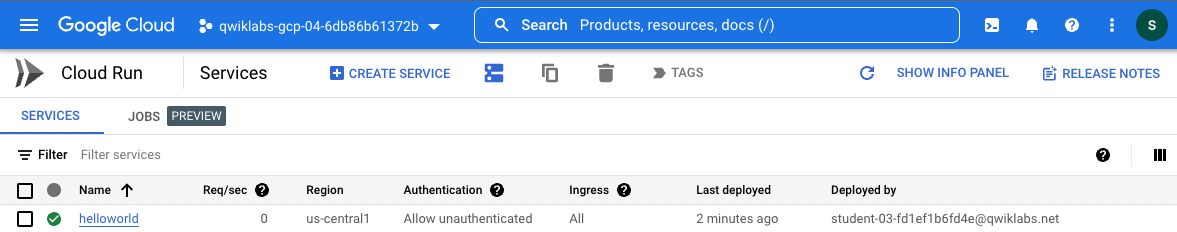
Service URL: https://helloworld-h6cp412q3a-uc.a.run.app

You can now visit your deployed container by opening the service URL in any browser window.

**Congratulations!** You have just deployed an application packaged in a container image to Cloud Run. Cloud Run automatically and horizontally scales your container image to handle the received requests, then scales down when demand decreases. In your own environment, you only pay for the CPU, memory, and networking consumed during request handling.

For this lab you used the gcloud command-line. Cloud Run is also available via Cloud Console.

* From the **Navigation menu**, in the Serverless section, click **Cloud Run** and you should see your helloworld service listed:



**Task 5. Clean up**

While Cloud Run does not charge when the service is not in use, you might still be charged for storing the built container image.

1. You can either decide to delete your GCP project to avoid incurring charges, which will stop billing for all the resources used within that project, or simply delete your helloworld image using this command :

gcloud container images delete gcr.io/$GOOGLE\_CLOUD\_PROJECT/helloworld

1. When prompted to continue type Y, and press **Enter**.
2. To delete the Cloud Run service, use this command :

gcloud run services delete helloworld --region="REGION"

1. When prompted to continue type Y, and press **Enter**.

**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.

**Congratulations!**

You have completed this lab!

Next steps / learn more

For more information on building a stateless HTTP container suitable for Cloud Run from code source and pushing it to Container Registry, view:

* [Developing Cloud Run services](https://cloud.google.com/run/docs/developing)
* [Building Containers](https://cloud.google.com/run/docs/building/containers)

**Manual Last Updated December 12, 2023**

**Lab Last Tested December 12, 2023**

Copyright 2022 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.

* [Overview](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step1)
* [Setup and requirements](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step2)
* [Task 1. Enable the Cloud Run API and configure your Shell environment](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step3)
* [Task 2. Write the sample application](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step4)
* [Task 3. Containerize your app and upload it to Artifact Registry](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step5)
* [Task 4. Deploy to Cloud Run](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step6)
* [Task 5. Clean up](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step7)
* [End your lab](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step8)
* [Congratulations!](https://googlecloud.qwiklabs.com/classrooms/21524/labs/191140#step9)

task\_alt

**Score Details**

OK

**How satisfied are you with this lab?\***

Top of Form

Additional Comments

Bottom of Form

Cancel

Submit

error\_outline

**Are you sure? You may not be able to restart the lab, and you'll need to start from the beginning if you do.**

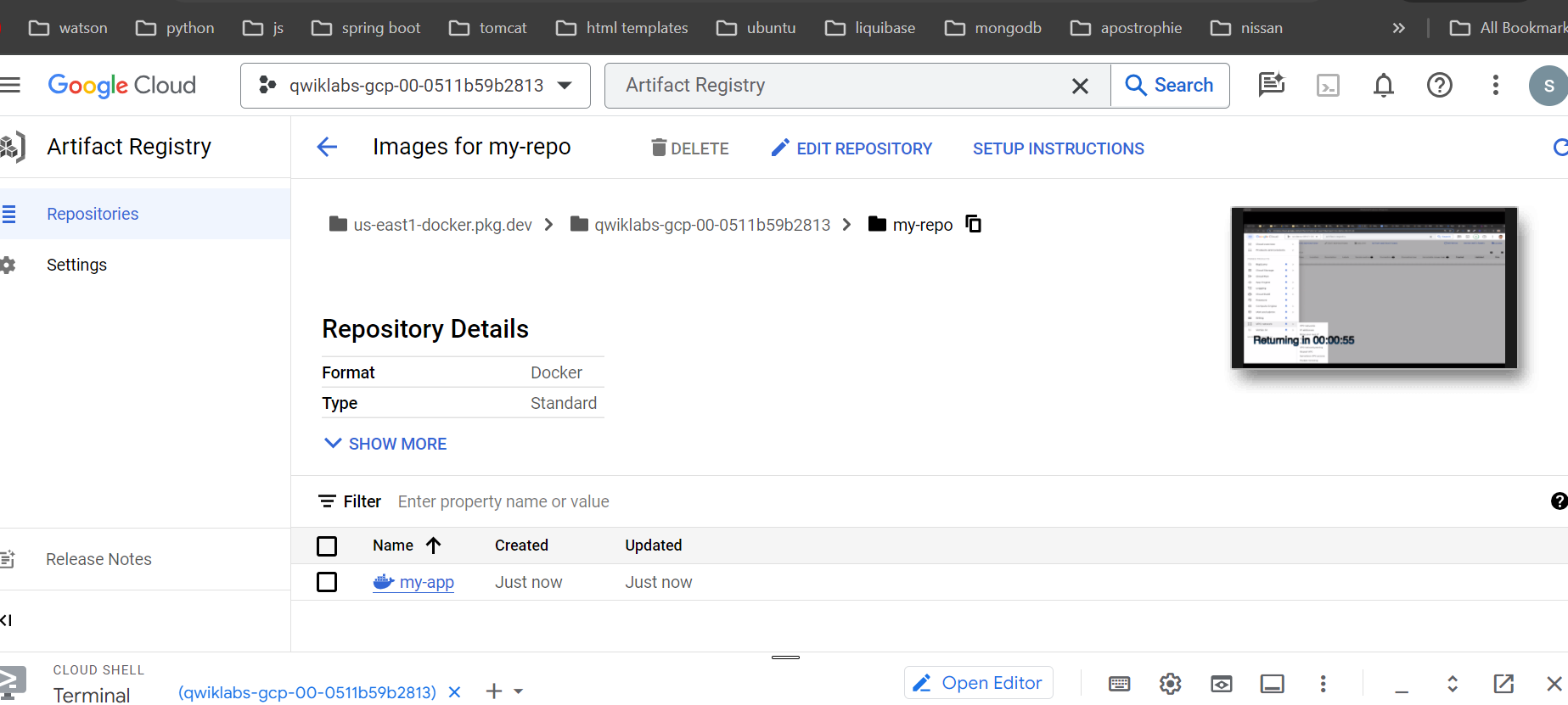
Cancel

End Lab

Docker

Build-packs -> builds light weigh containers

Cloud functions are similar to Lambda



# Creating and Running Docker Containers

1 hour1 Credit

## Overview

Docker is an open platform for developing, shipping, and running applications in containers. Docker helps you build, test, and deploy code faster, and it shortens the cycle between developing and running code. Docker does this by combining kernel containerization features with workflows and tooling that helps you manage and deploy your applications.

Docker lets you express the application build process by using a script, called a Dockerfile. Dockerfiles provide a low-level approach that offers flexibility at the cost of complexity. The Dockerfile is a manifest that details how to turn your source code into a container image.

Docker containers can be directly used in Cloud Run and Kubernetes, which allows them to be run on these platforms with ease. After learning the essentials of Docker, you will have the skill set to start developing containerized applications.

## Objectives

In this lab, you learn how to:

* Build, run, and debug Docker containers.
* Push Docker images to Artifact Registry, Google Cloud's container image repository.
* Pull Docker images from Artifact Registry.

## Setup

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

1. Sign in to Qwiklabs using an **incognito window**.
2. 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.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. 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**.
7. Accept the terms and skip the recovery resource page.

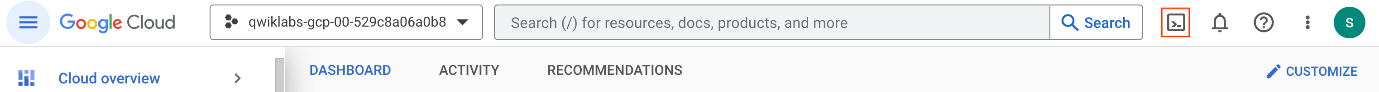
**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

### Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

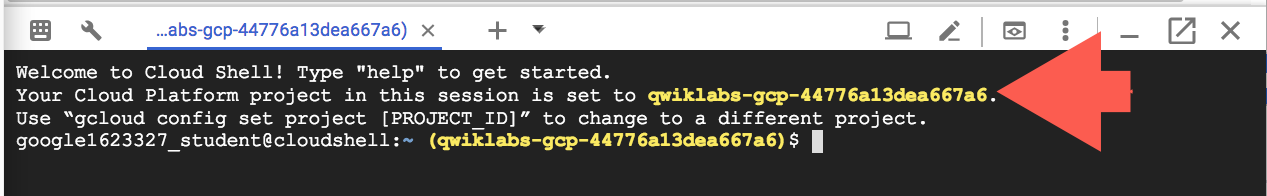
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Task 1. Set up your environment

You execute the shell commands in this lab in a separate VM that has been pre-provisioned for the lab.

1. To start an SSH session on the VM, in **Cloud Shell**, execute the following command:

gcloud compute ssh lab-vm --zone=us-east1-c

Copied!

content\_copy

1. When prompted, type **Y** to continue.
2. For the passphrase, type **Enter** to not use any.
3. Type **Enter** again.
4. Grant permissions to the student user on the socket used by Docker:

sudo chmod 666 /var/run/docker.sock

Copied!

content\_copy

1. To set your Project ID and region environment variables, run the following commands:
2. PROJECT\_ID=qwiklabs-gcp-00-0511b59b2813

REGION=us-east1

Copied!

content\_copy

Ensure that you run all the lab commands in the tasks below in the SSH session in Cloud Shell.

## Task 2. Build a container image

In this task, you create a Docker container image by using a Dockerfile.

### Create a Dockerfile

1. Create a test directory and change to it:

mkdir test && cd test

Copied!

content\_copy

1. Create the Dockerfile:
2. cat > Dockerfile <<EOF
3. # Use an official Node runtime as the parent image
4. FROM node:lts
5. # Set the working directory in the container to /app
6. WORKDIR /app
7. # Copy the current directory contents into the container at /app
8. COPY . /app
9. # Make the container's port 80 available to the outside world
10. EXPOSE 80
11. # Run app.js using node when the container launches
12. CMD ["node", "app.js"]

EOF

Copied!

content\_copy

**Note:**In this step, you assemble the container image by first selecting a parent or base image.

This is specified with the *FROM* instruction and the official Docker image for Node version lts (long term support).

The *WORKDIR* instruction sets the working directory in the container for any additional instructions that follow in the Dockerfile. For this lab, we use the /app directory as the container's working directory.

The *COPY* instruction copies directories or files from the source location to the destination path of the container image file system. Here, we copy files from the current directory to /app.

The *EXPOSE* instruction exposes the container's port so that it accepts connections on that port, which in this lab is port 80.

Finally, the *CMD* instruction provides the node command to execute the application in the running container.

### Develop your application

Next, you write the code for your Node.js application. This application is a simple HTTP server that listens for requests on port 80, and responds with a static message.

1. Create the app.js file with your application source code:
2. cat > app.js <<EOF
3. const http = require('http');
4. const hostname = '0.0.0.0';
5. const port = 80;
6. const server = http.createServer((req, res) => {
7. res.statusCode = 200;
8. res.setHeader('Content-Type', 'text/plain');
9. res.end('Welcome to your first Docker container!\n');
10. });
11. server.listen(port, hostname, () => {
12. console.log('Server running at http://%s:%s/', hostname, port);
13. });
14. process.on('SIGINT', function() {
15. console.log('Caught interrupt signal and will exit');
16. process.exit();
17. });

EOF

Copied!

content\_copy

**Note:** The application also implements the SIGINT handler function to log a message and exit gracefully when the container is stopped.

1. Verify the contents of the application source file:

cat app.js

Copied!

content\_copy

### Build the container image

In this subtask, you build the container image from the Dockerfile by using the docker build command.

1. Run the docker build command:

docker build -t my-app:0.1 .

Copied!

content\_copy

A partial command output is below:

=> [internal] load dockerignore

=> => transferring context: 2B

=> [internal] load build definition from Dockerfile

=> => transferring dockerfile: 394B

=> [internal] load metadata for docker.io/library/node:lts

=> [1/3] FROM docker.io/library/node:lts@sha256:586cdef48f920dea2f47a954b8717601933aa1daa0a08264abf9144789abf8ae

=> => resolve docker.io/library/node:lts@sha256:586cdef48f920dea2f47a954b8717601933aa1daa0a08264abf9144789abf8ae

=> => sha256:b7483c70b94e9fbb68e91d64456ee147d120488f876d69efeae815ba164e8b54 2.21kB / 2.21kB

...

...

=> [internal] load build context

=> => transferring context: 912B

=> [2/3] WORKDIR /app

=> [3/3] COPY . /app

=> exporting to image

=> => exporting layers

=> => writing image sha256:8cf51a1aba351cf505cd6d8eefa966b

=> => naming to docker.io/library/my-app:0.1

**Note:**This command builds the container image from the instructions in the Dockerfile in the current directory and tags the resulting image with the specified name and version. Google recommends to specify a tag to distinguish newer image versions from older ones.

1. View the list of images that were built:

docker images

Copied!

content\_copy

REPOSITORY TAG IMAGE ID CREATED SIZE

my-app 0.1 8cf51a1aba35 10 minutes ago 997MB

**Note:**The command lists the base *node* image and the *my-app* image that you built. The size of the image is relatively small compared to other virtual machine images.

To verify the objective, click **Check my progress**.

Assessment Completed!

Build a container image

Check my progress

*Assessment Completed!*

## Task 3. Run and test the application

After the container image is successfully built, you can run your application and test it to ensure that it behaves as expected.

### Run the container

1. To run the container, execute the command:

docker run -p 8080:80 --name my-app -d my-app:0.1

Copied!

content\_copy

**Note:**The *docker run* command starts the container with the specified name. The -p argument maps the host's port 8080 to the container's port 80, enabling requests to reach the server at http://localhost:8080. The -d argument runs the container in the background.

1. To view a list of running containers, execute the command:

docker ps

Copied!

content\_copy

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

46c209a1fd87 my-app:0.1 "docker-entrypoint.s…" 19 seconds ago Up 17 seconds 0.0.0.0:8080->80/tcp my-app

### Test the application

1. To test the application, send it a HTTP request by using the curl command:

curl http://localhost:8080

Copied!

content\_copy

1. Verify the application response:

Welcome to your first Docker container!

1. Stop the container by executing the docker stop command:

docker stop [CONTAINER ID]

Copied!

content\_copy

Replace [CONTAINER ID] with the value of the CONTAINER ID from the output of the command that was executed in the previous subtask.

## Task 4. Modify the container

You can have multiple versions of your containerized applications that share common layers in the image. In this task, you create another version of the containerized application and test both versions.

### Modify the application code

1. In the test directory, update the content of the app.js file:
2. cat > app.js <<EOF
3. const http = require('http');
4. const url = require('url');
5. const hostname = '0.0.0.0';
6. const port = 80;
7. const server = http.createServer((req, res) => {
8. res.statusCode = 200;
9. res.setHeader('Content-Type', 'text/plain');
10. console.log('Received request with URL: ?%s', req.url);
11. var q = url.parse(req.url, true).query;
12. res.end(q.user + ', Welcome to your first Docker container!\n');
13. });
14. server.listen(port, hostname, () => {
15. console.log('Server running at http://%s:%s/', hostname, port);
16. });
17. process.on('SIGINT', function() {
18. console.log('Caught interrupt signal and will exit');
19. process.exit();
20. });

EOF

Copied!

content\_copy

**Note:**The modified application code adds the value of a *user* query parameter that is passed in the HTTP request in the application response.

1. Verify the contents of the modified application source file with the cat command:

cat app.js

Copied!

content\_copy

### Rebuild the container image

1. Rebuild the image with a new tag:

docker build -t my-app:0.2 .

Copied!

content\_copy

1. Verify the output of the docker build command:
2. => [internal] load build definition from Dockerfile
3. => => transferring dockerfile: 394B
4. => [internal] load .dockerignore
5. => => transferring context: 2B
6. => [internal] load metadata for docker.io/library/node:lts
7. => [1/3] FROM docker.io/library/node:lts@sha256:586cdef48f920dea2f47a954b8717601933aa1daa0a08264abf9144789abf8ae
8. => [internal] load build context
9. => => transferring context: 691B
10. => CACHED [2/3] WORKDIR /app
11. => [3/3] COPY . /app
12. => exporting to image
13. => => exporting layers
14. => => writing image sha256:5fc2d7a43c4678da17daf204ef4b071f2da869ead758864622d90d880a40c24b

=> => naming to docker.io/library/my-app:0.2

As seen in the output, in the new container image only layers from Step 3 and below are modified because changes were made in the app.js source file.

### Run the new container

1. To run the new container, execute the command:

docker run -p 8080:80 --name my-app-2 -d my-app:0.2

Copied!

content\_copy

1. To view a list of running containers, execute the command:

docker ps

Copied!

content\_copy

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

7071749fe0f6 my-app:0.2 "docker-entrypoint.s…" 10 seconds ago Up 10 seconds 0.0.0.0:8081->80/tcp my-app-2

### Test the application

1. To test the application, send it a HTTP request by using curl:

curl http://localhost:8080?user=Learner

Copied!

content\_copy

1. Verify the application response:

Learner, Welcome to your first Docker container!

**Note:**The response contains the value of the *user* argument that was passed in the request to the container that is running the image version 0.2 of the application.

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

Assessment Completed!

Modify the container and rebuild the container image.

Check my progress

*Assessment Completed!*

## Task 5. Troubleshooting

There are some easy techniques to troubleshoot your containerized applications. We review some of these methods in this task.

### View container logs

1. First, retrieve the ID of the container whose logs you need to view:

docker ps

Copied!

content\_copy

1. To view the container logs, run the docker logs command:

docker logs [CONTAINER ID]

Copied!

content\_copy

Replace the CONTAINER ID with the initial characters of the container ID that uniquely identifies it from the output of the previous command. To tail the log's output as the container is running, use the -f option with the *logs* command.

Server running at http://0.0.0.0:80/

Received request with URL: ?/?user=Learner

### Start a shell inside the container

You can start an interactive shell inside a running container to troubleshoot it.

1. To start an interactive Bash session, run:

docker exec -it [CONTAINER ID] bash

Copied!

content\_copy

root@7071749fe0f6:/app#

1. From within this shell, to troubleshoot issues, you can inspect the container file system and other data files that your application might use.

ls

Copied!

content\_copy

1. Exit the Bash shell:

exit

Copied!

content\_copy

### Examine container metadata

1. To view a container's metadata, run:

docker inspect [CONTAINER ID]

Copied!

content\_copy

Here's a partial output from the command:

[

{

"Id": "7071749fe0f66b8b1953bbb6f28f159bd5dbeae079595675e2591d32d87ae5dc",

"Created": "2023-02-23T18:08:34.286519913Z",

"Path": "docker-entrypoint.sh",

"Args": [

"node",

"app.js"

],

"State": {

"Status": "running",

"Running": true,

"Paused": false,

"Restarting": false,

"OOMKilled": false,

"Dead": false,

"Pid": 1099,

"ExitCode": 0,

"Error": "",

"StartedAt": "2023-02-23T18:08:34.785098365Z",

"FinishedAt": "0001-01-01T00:00:00Z"

},

"Image": "sha256:5fc2d7a43c4678da17daf204ef4b071f2da869ead758864622d90d880a40c24b",

"ResolvConfPath": "/var/lib/docker/containers/7071749fe0f66b8b1953bbb6f28f159bd5dbeae079595675e2591d32d87ae5dc/resolv.conf",

"HostnamePath": "/var/lib/docker/containers/

...

...

}

]

**Note:**By default, the *inspect* command provides detailed metadata information in a JSON array. You can filter the results with the *--format* argument to inspect specific fields in the output.

## Task 6. Publishing a container image

Artifact Registry is a Google Cloud service that is used to store and manage software artifacts in private repositories, including container images, and software packages.

In this task, you push your container images to Artifact Registry making them available for deployment to other environments such as staging, and production, which make up your software delivery lifecycle.

### Create an image repository

Before you can push any container images to Artifact Registry, you must first create a repository.

1. In the Google Cloud console, in the **Navigation menu** (navigation menu), under **CI/CD**, navigate to **Artifact Registry > Repositories**.
2. In the **Create repository** page, provide the following information, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** (type or select) |
| **Name** | **my-repo** |
| **Format** | **Docker** |
| **Location type** | **Region** |
| **Region** | **us-east1** |

1. Click **Create**.

### Authenticate Docker to use the repository

Before you can push or pull images to or from the repository, you must configure Docker to authenticate requests to the repository in Artifact Registry.

1. To set up authentication to Docker repositories in the region us-east1, in the VM shell, run the following command:

gcloud auth configure-docker ${REGION}-docker.pkg.dev

Copied!

content\_copy

1. When prompted, type **Y**.

The full name of the repository that you created is: us-east1-docker.pkg.dev/qwiklabs-gcp-00-0511b59b2813/my-repo.

Docker image repository names use the format: *[location]-docker.pkg.dev* in Artifact Registry.

### Push the container to Artifact Registry

1. To push a container image to your private registry hosted by Artifact Registry, you need to first tag the image with the repository name:

docker build -t ${REGION}-docker.pkg.dev/${PROJECT\_ID}/my-repo/my-app:0.2 .

Copied!

content\_copy

1. List the Docker images that you built:

docker images

Copied!

content\_copy

REPOSITORY TAG IMAGE ID CREATED SIZE

my-app 0.2 9b1ef4854d32 4 minutes ago 997MB

us-east1-docker.pkg.dev/qwiklabs-gcp-02-7c092125ce3a/my-repo/my-app 0.2 9b1ef4854d32 4 minutes ago 997MB

my-app 0.1 8cf51a1aba35 5 minutes ago 997MB

1. To push the image to Artifact Registry, run the following command:

docker push ${REGION}-docker.pkg.dev/$PROJECT\_ID/my-repo/my-app:0.2

Copied!

content\_copy

The output of this command is similar to:

The push refers to repository [east1-docker.pkg.dev/qwiklabs-gcp-02-7c092125ce3a/my-repo/my-app]

b29bce04ddbb: Pushed

9ba0e19073ee: Pushed

3c397285cb7e: Pushed

a8d01c684adc: Pushed

56c4ec92f013: Pushed

4c92897e605e: Pushed

0b6859e9fff1: Pushed

11829b3be9c0: Pushed

dc8e1d8b53e9: Pushed

9d49e0bc68a4: Pushed

8e396a1aad50: Pushed

0.2: digest: sha256:383ffb5213f92e33dedb49042c0f070a9f76f263621226de20499dffd863b3df size: 2628

1. After the command completes, in the Google Cloud console, in the **Navigation menu** (navigation menu), under **CI/CD**, navigate to **Artifact Registry > Repositories**.
2. Click the **my-repo** repository to display the **my-app** Docker container image.

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

Assessment Completed!

Publish a container image.

Check my progress

*Assessment Completed!*

## Task 7. Pull and test the container image

In this task, you start with a fresh environment and then pull the container image from Artifact Registry to test it. To simulate another environment, you stop and remove all containers and images from your shell environment that were created in the previous tasks in this lab.

### Stop and remove containers

1. To stop all running containers, execute the following command:

docker stop $(docker ps -q)

Copied!

content\_copy

1. To remove all containers, execute the following command:

docker rm $(docker ps -aq)

Copied!

content\_copy

### Remove all container images

1. To remove the registry-tagged container image, execute the following command:

docker rmi ${REGION}-docker.pkg.dev/$PROJECT\_ID/my-repo/my-app:0.2

Copied!

content\_copy

**Note:**This command does not remove the container image from the registry.

1. To remove all other images, execute the following command:

docker rmi -f $(docker images -aq)

Copied!

content\_copy

1. Verify that no container images exist in your VM environment:

docker images

Copied!

content\_copy

REPOSITORY TAG IMAGE ID CREATED SIZE

You should now have a fresh host environment without any local images.

### Test the image

Test the image by pulling it from Artifact Registry.

1. To pull the image from Artifact Registry, execute the following command:

docker pull ${REGION}-docker.pkg.dev/${PROJECT\_ID}/my-repo/my-app:0.2

Copied!

content\_copy

1. To list the image, execute the following command:

docker images

Copied!

content\_copy

1. To run the container, execute the following command:

docker run -p 8080:80 -d ${REGION}-docker.pkg.dev/${PROJECT\_ID}/my-repo/my-app:0.2

Copied!

content\_copy

1. To test the application, run:

curl http://localhost:8080?user=Learner

Copied!

content\_copy

**Note:**This task demonstrates container portability where you can run containerized applications on other VMs or environments with Docker, and without the need to install any application dependencies on the host machine. The container images can be hosted on public or private registries which should be accessible by Docker.

## Congratulations!

Congratulations on completing this lab on the fundamentals of creating containers with Docker. In this lab, you:

* Built container images with Docker and ran Docker containers.
* Pushed Docker images to Artifact Registry, Google Cloud's container image repository.
* Pulled Docker images from Artifact Registry and ran them in a fresh environment to verify container portability.

### Next Steps / Learn More

For more information about Docker, and Artifact Registry, view the documentation:

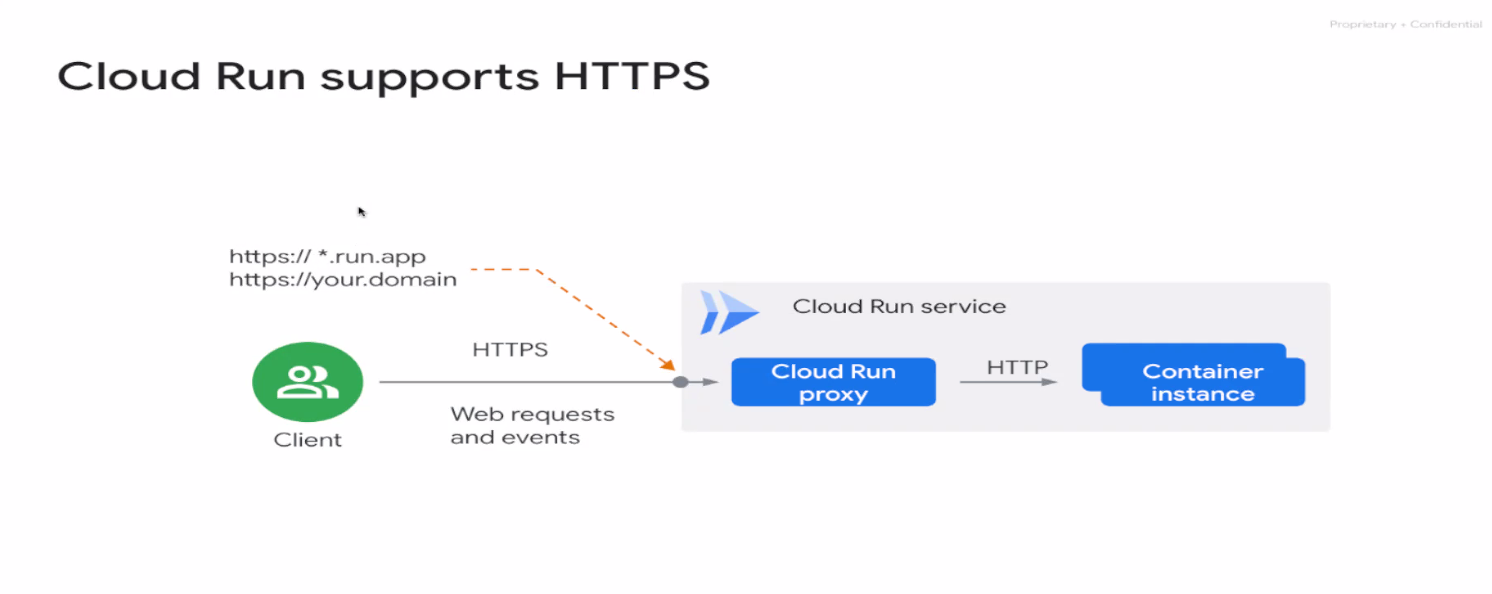
* [Docker documentation](https://www.docker.com/)
* [Docker command reference](https://docs.docker.com/reference/)
* [Artifact Registry](https://cloud.google.com/artifact-registry)

Copyright 2022 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.

* [Overview](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step1)
* [Objectives](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step2)
* [Setup](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step3)
* [Task 1. Set up your environment](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step4)
* [Task 2. Build a container image](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step5)
* [Task 3. Run and test the application](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step6)
* [Task 4. Modify the container](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step7)
* [Task 5. Troubleshooting](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step8)
* [Task 6. Publishing a container image](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step9)
* [Task 7. Pull and test the container image](https://googlecloud.qwiklabs.com/classrooms/21525/labs/191141#step10)

Cloud Run

# Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications

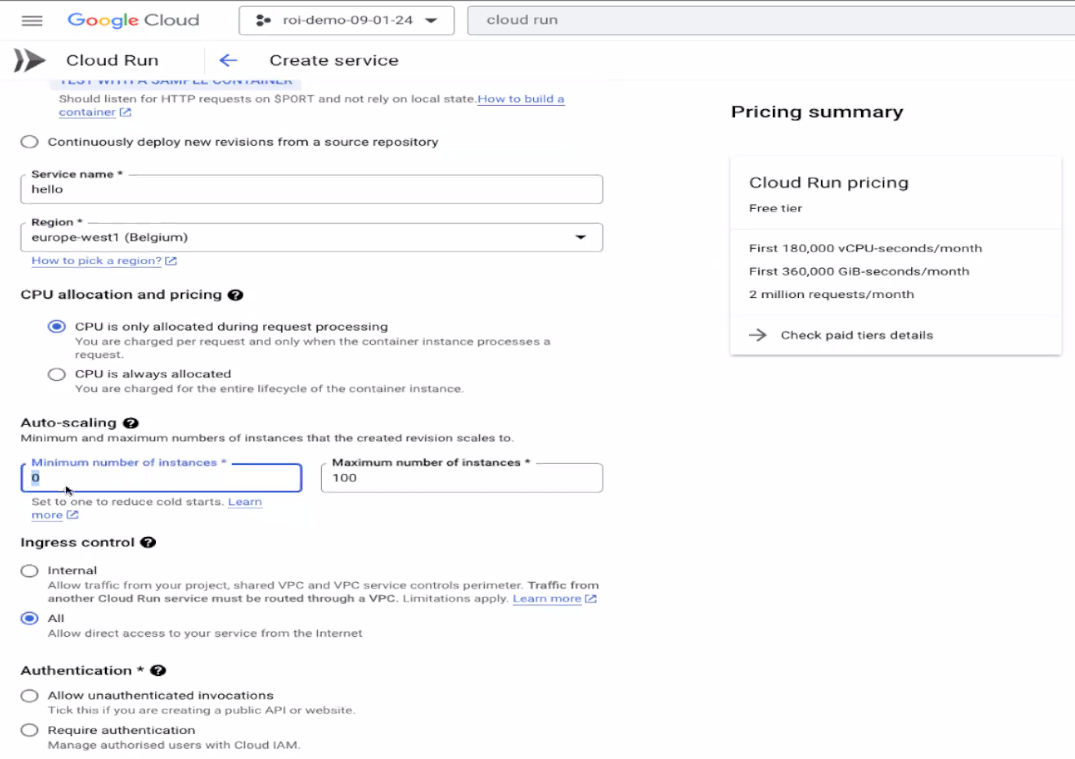


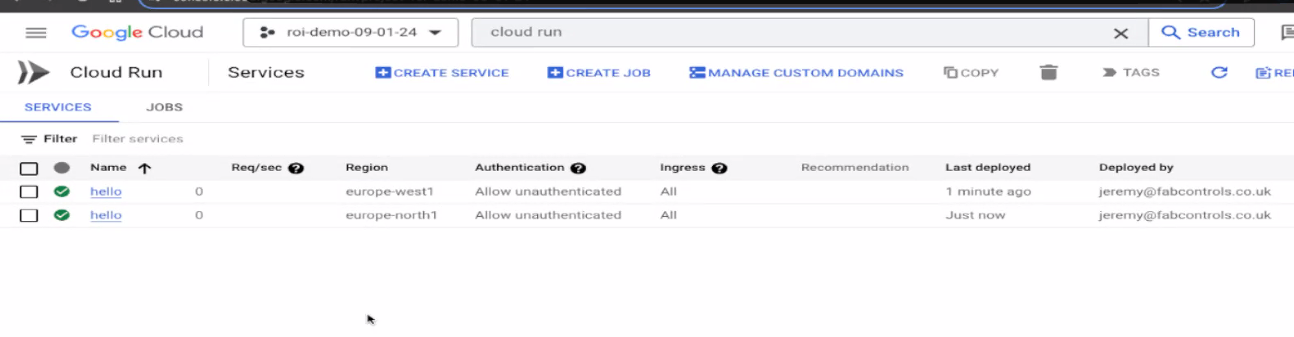
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated





A screenshot of a project

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a service

Description automatically generated

A close-up of a white background

Description automatically generated

A diagram of a service

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screen shot of a menu

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Deploying a Containerized Application on Cloud Run

45 minutes1 Credit

## Overview

[Cloud Run](https://cloud.google.com/run) is a fully managed compute platform that allows you to run stateless containers that are invocable with HTTP requests. You can deploy code written in any programming language on Cloud Run if you can build a container image from it. You can use the source-based deployment option that builds the container for you when developing your application in Go, Node.js, Python, Java, .NET Core, or Ruby.

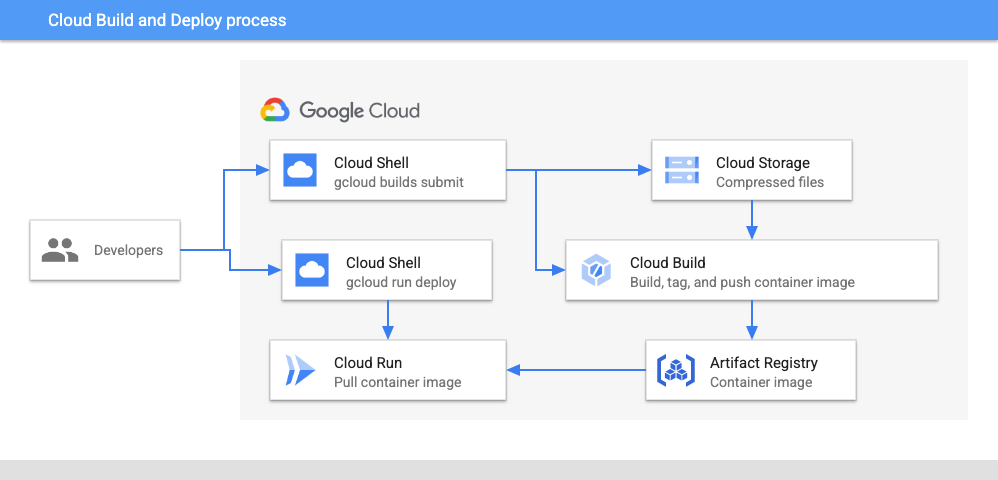
Cloud Run is serverless, and runs your containers on Google's scalable infrastructure. It's built on Knative, and lets you choose to run your containers either fully managed with Cloud Run, or in your Google Kubernetes Engine cluster with Cloud Run on GKE.

Cloud Run works well with other services on Google Cloud, so you can build full-featured applications without spending too much time operating, configuring, and scaling your Cloud Run service.

## Objectives

In this lab, you:

* Use Cloud Build to create a Docker container image for your application.
* Deploy the container image to Cloud Run.
* Run and test the containerized application.



## Setup

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

1. Sign in to Qwiklabs using an **incognito window**.
2. 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.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. 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**.
7. Accept the terms and skip the recovery resource page.

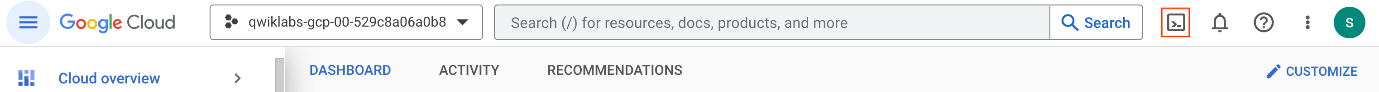
**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

### Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

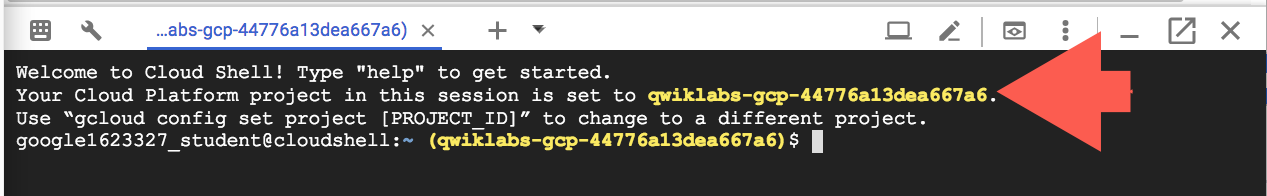
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Task 1. Configure your project and environment

In this task, you set environment variables for your Cloud Shell environment, and enable the relevant Google APIs for use in this lab.

1. Sign in to the Google Cloud console with your lab credentials, and open the **Cloud Shell** terminal window.
2. To set your project ID and region environment variables, in Cloud Shell, run the following commands:
3. PROJECT\_ID=$(gcloud config get-value project)

REGION=us-central1

Copied!

content\_copy

1. To enable relevant APIs, run the following command:
2. gcloud services enable artifactregistry.googleapis.com \
3. cloudbuild.googleapis.com \

run.googleapis.com

Copied!

content\_copy

To verify the objective, click **Check my progress**.

Enable relevant APIs.

Check my progress

## Task 2. Test the application locally

In this task, you copy a sample Node.js application. Then, you build and run it locally in your Cloud Shell environment.

### Copy the application

1. Create an app directory, and make it the current working directory:

mkdir app && cd app

Copied!

content\_copy

1. To copy the app from Cloud Storage, and extract it's files, execute the following command:

gsutil cp gs://cloud-training/CBL515/sample-apps/sample-node-app.zip . && unzip sample-node-app

Copied!

content\_copy

### Install application dependencies

1. Change to the app directory:

cd sample-node-app

Copied!

content\_copy

1. To view the application source code, view the contents of the server.js file in the top-level directory:

cat server.js

Copied!

content\_copy

This is the main application entry point. This sample application is a basic retail service that provides an API that returns product data when requests are made to the application over HTTP.

1. To install the app's dependency modules locally, execute the following command:

npm install

Copied!

content\_copy

### Test the application

1. To run the app locally, execute the following command:

npm start

Copied!

content\_copy

You should see the following output:

> sample-node-app@1.0.0 start

> node server.js

Sample-node-app listening on port 8080!

1. To open a second **Cloud Shell** terminal window, in the Cloud Shell navigation bar, click (Add).
2. Test the sample API by making a few HTTP requests to the application:

curl http://localhost:8080/service/products | jq

Copied!

content\_copy

We use the command line tool *jq* to parse and format the JSON response from the application.

The application should respond with a list of products:

[

{

"id": "1YMWWN1N4O",

"name": "Home Barista Kit",

"description": "Always wanted to brew coffee with Chemex and Aeropress at home?",

"picture": "static/img/products/barista-kit.jpg",

"cost": 124,

"categories": [

"cookware"

]

},

{

"id": "L9ECAV7KIM",

"name": "Terrarium",

"description": "This terrarium will look great in your white painted living room.",

"picture": "static/img/products/terrarium.jpg",

"cost": 36.45,

"categories": [

"gardening"

]

},

{

"id": "2ZYFJ3GM2N",

"name": "Film Camera",

"description": "This camera looks like it's a film camera, but it's actually digital.",

"picture": "static/img/products/film-camera.jpg",

"cost": 2245,

"categories": [

"photography",

"vintage"

]

},

{

"id": "LS4PSXUNUM",

"name": "Metal Camping Mug",

"description": "You probably don't go camping that often but this is better than plastic cups.",

"picture": "static/img/products/camp-mug.jpg",

"cost": 24.33,

"categories": [

"cookware"

]

},

{

"id": "9SIQT8TOJO",

"name": "City Bike",

"description": "This single-gear bike probably cannot climb the hills of San Francisco.",

"picture": "static/img/products/city-bike.jpg",

"cost": 789.5,

"categories": [

"cycling"

]

},

{

"id": "6E92ZMYYFZ",

"name": "Air Plant",

"description": "Have you ever wondered whether air plants need water? Buy one and figure out.",

"picture": "static/img/products/air-plant.jpg",

"cost": 12.3,

"categories": [

"gardening"

]

}

]

1. Make a second API request to fetch a specific product by ID:

curl http://localhost:8080/service/products/1YMWWN1N4O | jq

Copied!

content\_copy

The application should respond with details about the specific product:

{

"id": "1YMWWN1N4O",

"name": "Home Barista Kit",

"description": "Always wanted to brew coffee with Chemex and Aeropress at home?",

"picture": "static/img/products/barista-kit.jpg",

"cost": 124,

"categories": [

"cookware"

]

}

1. To exit the application, in the first Cloud Shell terminal window, type Ctrl-C.

## Task 3. Containerize the application with Cloud Build

Cloud Build is a service that executes your builds on Google Cloud. With Cloud Build, you can continuously build, test, and deploy your application by using a continuous integration and delivery (CI/CD) pipeline.

To provide instructions to Cloud Build, you create a build configuration file that contains a set of tasks. These instructions can configure builds to fetch dependencies, run unit and integration tests, perform static analyses, and create artifacts with build tools like docker, gradle, maven, and others.

In this task, you use Cloud Build to build a Docker container for your application, and push the resulting container image to a repository in Artifact Registry.

### Create the repository

1. In the Google Cloud console, in the navigation menu (navigation menu) under **CI/CD**, navigate to **Artifact Registry > Repositories**.
2. Click **Create Repository**.
3. In the **Create repository** page, provide the following information, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** (type or select) |
| **Name** | **my-repo** |
| **Format** | **Docker** |
| **Location type** | **Region** |
| **Region** | **us-central1** |

1. Click **Create**, and wait for the repository to be created.

### Authenticate Docker to use the repository

Before you can push images to the repository, you must configure Docker to authenticate requests to the repository in Artifact Registry.

1. To set up authentication to Docker repositories in the region us-central1, in Cloud Shell, run the following command:

gcloud auth configure-docker ${REGION}-docker.pkg.dev

Copied!

content\_copy

1. When prompted, type **Y**.

### Create the build configuration file

To provide instructions to Cloud Build, you create a build configuration file that contains a set of tasks. These instructions can configure builds to fetch dependencies, run unit and integration tests, perform static analyses, and create artifacts with builders like docker, gradle, maven, and others.

1. Set an environment variable for the repository name:

REPO=${REGION}-docker.pkg.dev/${PROJECT\_ID}/my-repo

Copied!

content\_copy

1. Create the build configuration file:
2. cat > cloudbuild.yaml <<EOF
3. steps:
4. - name: 'gcr.io/cloud-builders/docker'
5. args: [ 'build', '-t', '${REPO}/sample-node-app-image', '.' ]
6. images:
7. - '${REPO}/sample-node-app-image'

EOF

Copied!

content\_copy

1. View the cloudbuild.yaml file:

cat cloudbuild.yaml

Copied!

content\_copy

The build step builds a container image by using the docker builder from source code and the Dockerfile which are located in the current directory. The built image is then pushed to the repository in Artifact Registry.

### Build the container

1. To build the container with **Cloud Build**, execute the following command:

gcloud builds submit --region=$REGION --config=cloudbuild.yaml

Copied!

content\_copy

Here's similar (partial) output from the command:

...

...

b3389e626b47: Pushed

38610c0cfc18: Pushed

latest: digest: sha256:b6007afa5e8fb05d8ac617ddf5fee8d58cc6ba8901038c97f8e360520c5fdbf4 size: 3051

DONE

----------------------------------------------------------------------------------------------------------------------------------------

ID: f803a828-dc50-41cb-bc94-71dfb8c83648

CREATE\_TIME: 2023-02-28T18:28:41+00:00

DURATION: 1M23S

SOURCE: gs://qwiklabs-gcp-02-26de4317fac8\_cloudbuild/source/1677608919.352478-c157fa4ae8544b3dbba23b5f29145286.tgz

IMAGES: asia-east1-docker.pkg.dev/qwiklabs-gcp-02-26de4317fac8/my-repo/sample-node-app-image (+1 more)

STATUS: SUCCESS

Cloud Build first uploads your application source code and other files from the specified directory to Cloud Storage. It then builds the container image that contains your application from the instructions specified in the build configuration and Dockerfile, tags the image with the specified image name, and pushes the image to the repository in Artifact Registry.

### View build history

1. In the Google Cloud console, in the navigation menu (navigation menu) under **CI/CD**, click **Cloud Build > History**.
2. For **Region**, select us-central1.
3. Click the build ID for the build at the top of the list.

The details of the build, including the build log, are displayed.

### View the container image in Artifact Registry

1. After the build command completes, in the Google Cloud console, in the navigation menu (navigation menu) under **CI/CD**, navigate to **Artifact Registry > Repositories**.
2. Click the **my-repo** repository to display the **sample-node-app-image** container image.

To verify the objective, click **Check my progress**.

Build a container image with Cloud Build.

Check my progress

## Task 4. Deploy the container to Cloud Run

With the container image built, you can now deploy it to Cloud Run. There are two approaches for deploying to Cloud Run:

* [Managed Cloud Run](https://cloud.google.com/run): A fully managed service model where the entire container lifecycle is managed by Cloud Run. You use this approach in this lab.
* [Cloud Run on Anthos](https://cloud.google.com/run/docs/gke/setup): Cloud Run with an extra layer of control, which lets you bring your own clusters and pods from GKE.

In this task, you deploy your container image to the fully managed Cloud Run service on Google Cloud.

1. To deploy the container image, in Cloud Shell, execute the following command:

gcloud run deploy sample-node-app --image ${REPO}/sample-node-app-image --region $REGION --allow-unauthenticated

Copied!

content\_copy

The *allow-unauthenticated* option enables access to the service without requiring any authentication.

1. After the command completes, in the Google Cloud console, in the navigation menu (navigation menu) click **Cloud Run**.
2. To display the details of the Cloud Run service, click the **sample-node-app** service name.

To verify the objective, Click **Check my progress**.

Deploy the container image to Cloud Run.

Check my progress

## Task 5. Test the application on Cloud Run

1. To verify that the service is running, and available to accept requests, in Cloud Shell, execute the following command:

gcloud run services list

Copied!

content\_copy

The output of the command is similar to:

✔

SERVICE: sample-node-app

REGION: asia-east1

URL: https://sample-node-app-dduno3adrq-uc.a.run.app

LAST DEPLOYED BY: student-04-329a97f025fd@qwiklabs.net

LAST DEPLOYED AT: 2023-02-28T22:21:04.803928Z

1. From the command output, copy the value of the URL, and paste it in the curl command appending service/products to the end of the URL as shown. Pipe the output to the jq command, and press ENTER:

curl https://sample-node-app-dduno3adrq-uc.a.run.app/service/products | jq

The output should be similar to that received from the application when you ran it locally in an earlier task.

To verify the objective, click **Check my progress**.

Test the application that is deployed on Cloud Run.

Check my progress

## Congratulations!

In this lab, you:

* Used Cloud Build to create a Docker container image for your application, and deployed the container image to Cloud Run.
* You also tested the application locally and on Cloud Run to verify that it works as expected.

### Next Steps / Learn More

For more information, view the documentation on:

* [Cloud Build](https://cloud.google.com/build/docs/overview)
* [Cloud Run](https://cloud.google.com/run/)

Copyright 2022 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.

A screenshot of a computer

Description automatically generated

# Creating a Containerized Application with Buildpacks

30 minutesNo cost

## Overview

Buildpacks are another approach for building container images and provide an alternate approach to turn your source code into a container image. Buildpacks are distributed and executed in images called builders. Each builder can have one or more buildpacks. A builder turns your source code into a container image. The buildpacks do the actual work to build and package the container image that you can deploy to Cloud Run or run with Docker locally.

You can create your own buildpacks, or use those provided by multiple vendors. Google Cloud's buildpacks allow developers to create and deploy containerized applications without the need to install Docker locally, or create a Dockerfile. Buildpacks are also built into Cloud Run to enable a source-based deployment workflow.

## Objectives

In this lab, you:

* Build an application with pack, a command-line tool that is used with builders to create container images from source code.
* Use the Google Cloud's buildpacks builder to build a container image.
* Run and test the container locally with Docker.
* Build and redeploy the container to Cloud Run.

## Setup

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

1. Sign in to Qwiklabs using an **incognito window**.
2. 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.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. 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**.
7. Accept the terms and skip the recovery resource page.

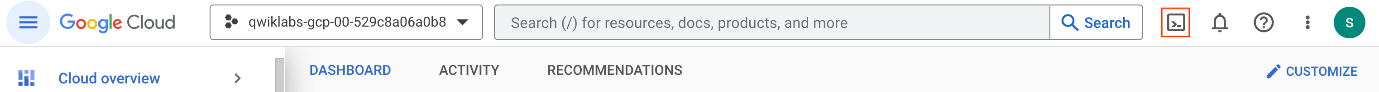
**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

### Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

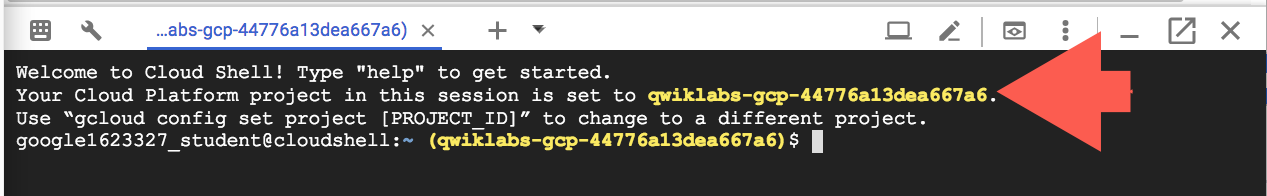
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Task 1. Configure your environment and project

In this task, you set environment variables, and configure your Cloud Shell environment.

### Configure your Cloud Shell environment

1. To set your project ID and region environment variables, in Cloud Shell, run the following commands:
2. PROJECT\_ID=$(gcloud config get-value project)

REGION=us-central1

Copied!

content\_copy

1. Set the compute region in Cloud Shell:

gcloud config set compute/region $REGION

Copied!

content\_copy

### Enable Google APIs

* To use Cloud Run, and the Google Translate API later in this lab, enable relevant APIs for your project:

gcloud services enable artifactregistry.googleapis.com run.googleapis.com translate.googleapis.com

Copied!

content\_copy

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

Enabled Google APIs

Check my progress

## Task 2. Build and run an application with Docker

In this task, you build a sample application with the pack command line tool and the Google Cloud's buildpacks builder.

### Develop the application

1. Create an app directory and change to it:

mkdir app && cd app

Copied!

content\_copy

1. Copy the sample python application for this lab from Cloud Storage, and extract the contents from the archive:

gsutil cp gs://cloud-training/CBL513/sample-apps/sample-py-app.zip . && unzip sample-py-app

Copied!

content\_copy

1. View the sample application files and source code:

ls sample-py-app

Copied!

content\_copy

cat sample-py-app/main.py

Copied!

content\_copy

Because the Python buildpack does not generate a default container entry-point for the application, we use a *Procfile* to configure the application's start command.

The application is written in Python and returns a sample welcome message in response to a request made to the application.

### Build the container

1. Change to the sample application directory:

cd sample-py-app

Copied!

content\_copy

1. To build the container, run pack:

pack build --builder=gcr.io/buildpacks/builder sample-py-app

Copied!

content\_copy

A partial output is similar to:

...

...

[exporter] Setting default process type 'web'

[exporter] Saving sample-py-app...

[exporter] \*\*\* Images (9f9f9a48fd46):

[exporter] sample-py-app

[exporter] Adding cache layer 'google.python.pip:pip'

[exporter] Adding cache layer 'google.python.pip:pipcache'

Successfully built image sample-py-app

**Note:**With *pack*, you did not need to write and provide a *Dockerfile*to build the container image.

1. To view the images downloaded and built in your Cloud Shell host, run:

docker images

Copied!

content\_copy

REPOSITORY TAG IMAGE ID CREATED SIZE

gcr.io/buildpacks/builder latest 514fb6f1bbfe 29 hours ago 804MB

gcr.io/buildpacks/gcp/run v1 22db1b5e48e3 29 hours ago 177MB

buildpacksio/lifecycle 0.16.0 67e021546a3f 43 years ago 30.5MB

sample-py-app latest 9f9f9a48fd46 43 years ago 571MB

1. Run the container locally in Docker by passing in the PORT environment variable to the application and binding the host's port 8080 to the container port:

docker run -it -e PORT=8080 -p 8080:8080 -d sample-py-app

Copied!

content\_copy

The application code listens on the port that is provided in the environment variable, in this case, port 8080.

1. Test the containerized application with the curl command:

curl http://localhost:8080/

Copied!

content\_copy

You should see the following message as a response:

Welcome to this sample app, built with Buildpacks.

## Task 3. Build and run an application on Cloud Run

Typically, as a next step in your development and deployment lifecycle, you should push the container image that you built in the previous task to Artifact Registry, and then deploy the image to a container-based environment like Google Kubernetes Engine or Cloud Run.

In this task, you modify the sample application code, then build and deploy the containerized application directly from source with Cloud Run.

### Modify the application code

You modify the sample application code to use the Google Translation API that translates a piece of text from English to Spanish.

1. Edit the main.py file with an editor of your choice, for example, **vi** or **nano**. You can also click **Open Editor** from the Cloud Shell menu to edit the file.
2. Replace the entire contents of the main.py file with the code below:
3. from flask import Flask, request
4. import google.auth
5. from google.cloud import translate
6. app = Flask(\_\_name\_\_)
7. \_, PROJECT\_ID = google.auth.default()
8. TRANSLATE = translate.TranslationServiceClient()
9. PARENT = 'projects/{}'.format(PROJECT\_ID)
10. SOURCE, TARGET = ('en', 'English'), ('es', 'Spanish')
11. @app.route('/', methods=['GET', 'POST'])
12. def index():
13. # reset all variables
14. text = translated = None
15. if request.method == 'POST':
16. text = request.get\_json().get('text').strip()
17. if text:
18. data = {
19. 'contents': [text],
20. 'parent': PARENT,
21. 'target\_language\_code': TARGET[0],
22. }
23. # handle older call for backwards-compatibility
24. try:
25. rsp = TRANSLATE.translate\_text(request=data)
26. except TypeError:
27. rsp = TRANSLATE.translate\_text(\*\*data)
28. translated = rsp.translations[0].translated\_text
29. # create context
30. context = {
31. 'trtext': translated
32. }
33. return context
34. if \_\_name\_\_ == "\_\_main\_\_":
35. # Dev only: run "python main.py" and open http://localhost:8080
36. import os

app.run(host="localhost", port=int(os.environ.get('PORT', 8080)), debug=True)

Copied!

content\_copy

The application code uses the Google Translate API to translate a piece of text passed in a JSON request from English to Spanish.

### Build and deploy the container

1. To build and deploy the container on Cloud Run, execute the following command:

gcloud run deploy sample-py-app --source . --region=${REGION} --allow-unauthenticated

Copied!

content\_copy

The *allow-unauthenticated* option enables access to the service without requiring any authentication.

1. When prompted, type **Y** to accept the default repository that is created in Artifact Registry to store the container image.
2. When the command completes, a Cloud Run service named sample-py-app is created.

The command output is similar to:

Building using Buildpacks and deploying container to Cloud Run service [sample-py-app] in project [qwiklabs-gcp-00-0d56d42aca1a] region [asia-east1]

OK Building and deploying new service... Done.

OK Creating Container Repository...

OK Uploading sources...

OK Building Container... Logs are available at [https://console.cloud.google.com/cloud-build/builds/8bea2ded-4745-41f9-a82d-128e409daa20?project=34240880885].

OK Creating Revision...

OK Routing traffic...

OK Setting IAM Policy...

Done.

Service [sample-py-app] revision [sample-py-app-00001-nec] has been deployed and is serving 100 percent of traffic.

Service URL: https://sample-py-app-ulvp7xw3bq-de.a.run.app

### Test the Cloud Run service

1. Set an environment variable for the Cloud Run service that was created in the previous step:

SERVICE\_URL=[SERVICE URL]

Copied!

content\_copy

Replace the [SERVICE URL] with the value returned from Cloud Run in the output of the command in the previous step.

1. To test the service, , and execute the curl command:

curl $SERVICE\_URL -H 'Content-Type: application/json' -d '{"text" : "Welcome to this sample app, built with Google Cloud buildpacks."}'

Copied!

content\_copy

{"trtext":"Bienvenido a esta aplicaci\u00f3n de muestra, creada con paquetes de compilaci\u00f3n de Google Cloud."}

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

Deploy an application on Cloud Run

Check my progress

## Congratulations!

In this lab, you built an application with the pack command, and the Google Cloud's buildpacks builder to build a container image for a sample python application. You first ran and tested the container locally with Docker before rebuilding and deploying the container to Cloud Run.

### Next Steps / Learn More

For more information, view the documentation:

* [Google Cloud's buildpacks](https://cloud.google.com/docs/buildpacks/overview)
* [Cloud Run](https://cloud.google.com/run)

Learn about Google Cloud's buildpacks for different container platforms such as GKE, Anthos, and Container-Optimized OS:

* [Google Cloud's buildpacks github repository](https://github.com/GoogleCloudPlatform/buildpacks)

Copyright 2022 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.

Notes:

Cloud function gen 1 = 9 mins

Cloud functions gen2 = 60 mins

For long running work load cloud run jobs

Cloud run runs on Kubernetes cluster

Autopilot mode or standard Kubernetes cluster

# Implementing Least Privilege IAM Policy Bindings in Cloud Run [APPRUN]

30 minutes1 Credit

## Overview

The principle of least privilege states that a resource should only have access to the exact set of resources it needs in order to function. For example, if a service is performing an automated database backup, the service should be restricted to read-only permissions on exactly one database. Similarly, if a service is only responsible for encrypting data, it should not have permissions for decrypting data.

In Cloud Run, if a service is deployed without specifying a service account, a default service account is used. The default service account used is the Compute Engine service account which has the broad Editor role on the project. Because of policy binding inheritance, this service account has read and write permissions on most resources in your project. While convenient, it's an inherent security risk as resources can be created, modified, or deleted with this service account.

To mitigate this risk and implement the principle of least privilege, you should create a service account that serves as the service's identity, and grant the minimum set of permissions to the account that are required for the service's functionality.

### Objectives

In this lab, you learn to:

* Configure your environment and enable the Cloud Run API.
* Create and deploy a public Cloud Run service.
* Test the service with unauthenticated requests.
* Create a service account with minimum permissions.
* Use the gcloud CLI to authenticate with the service account, and invoke a Cloud Run service.
* Implement least privilege by granting the minimum set of permissions required to invoke a service on Cloud Run.

### Prerequisites

These labs are based on intermediate knowledge of Google Cloud. While the steps required are covered in the content, it would be helpful to have familiarity with any of the following products:

* IAM
* Cloud Run

## Setup and requirements

#### Before you click the Start Lab button

**Note: Read these instructions.**

Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

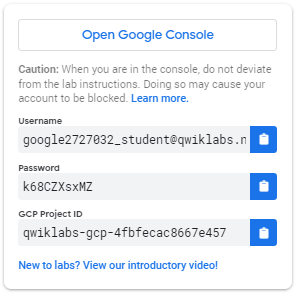
To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:**If you already have your own personal Google Cloud account or project, do not use it for this lab.**Note:**If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Console

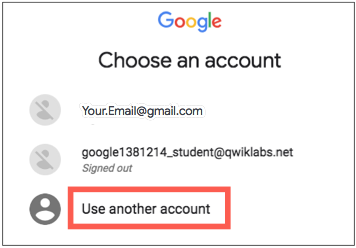
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

**Note:**Open the tabs in separate windows, side-by-side.

1. On the Choose an account page, click **Use Another Account**. The Sign in page opens.



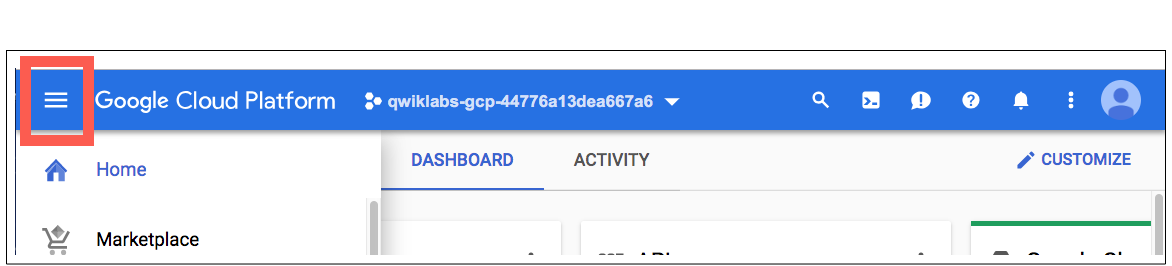
1. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

**Note:**You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

1. Click through the subsequent pages:

* Accept the terms and conditions.
* Do not add recovery options or two-factor authentication (because this is a temporary account).
* Do not sign up for free trials.

After a few moments, the Cloud console opens in this tab.

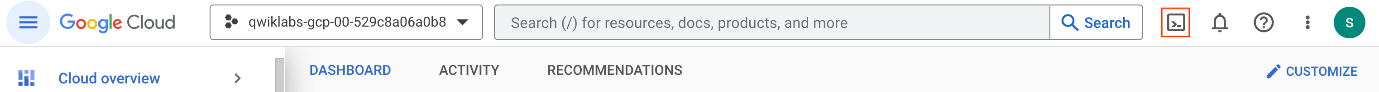
**Note:**You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

### Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

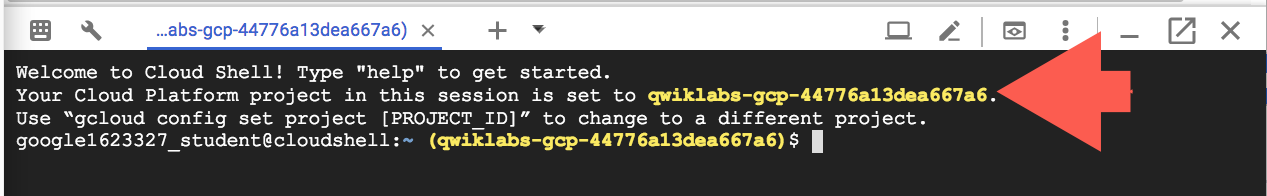
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Task 1. Configure the environment

Set up environment variables in Cloud Shell to make the provisioning process more flexible.

1. Enable Cloud Run API:

gcloud services enable run.googleapis.com

Copied!

content\_copy

1. Create a LOCATION environment variable:

LOCATION=us-east1

Copied!

content\_copy

1. Set the default Cloud Run region:

gcloud config set run/region $LOCATION

Copied!

content\_copy

## Task 2. Create and deploy a public service

### Requirements

Quickway parking has a Cloud Run billing service that they would like to be made more secure. In this task, you:

* Deploy the billing service from an image.
* Test the service by invoking it without any authentication.

### Deploying with Cloud Run

The Quickway development team already has an image of the billing application available on Google Cloud.

1. Deploy the billing application image to Cloud Run:
2. gcloud run deploy billing-service \
3. --image gcr.io/qwiklabs-resources/gsp723-parking-service \
4. --region $LOCATION \

--allow-unauthenticated

Copied!

content\_copy

1. Assign the URL of the new service to an environment variable:
2. BILLING\_SERVICE\_URL=$(gcloud run services list \
3. --format='value(URL)' \

--filter="billing-service")

Copied!

content\_copy

1. Invoke the service without any authorization:

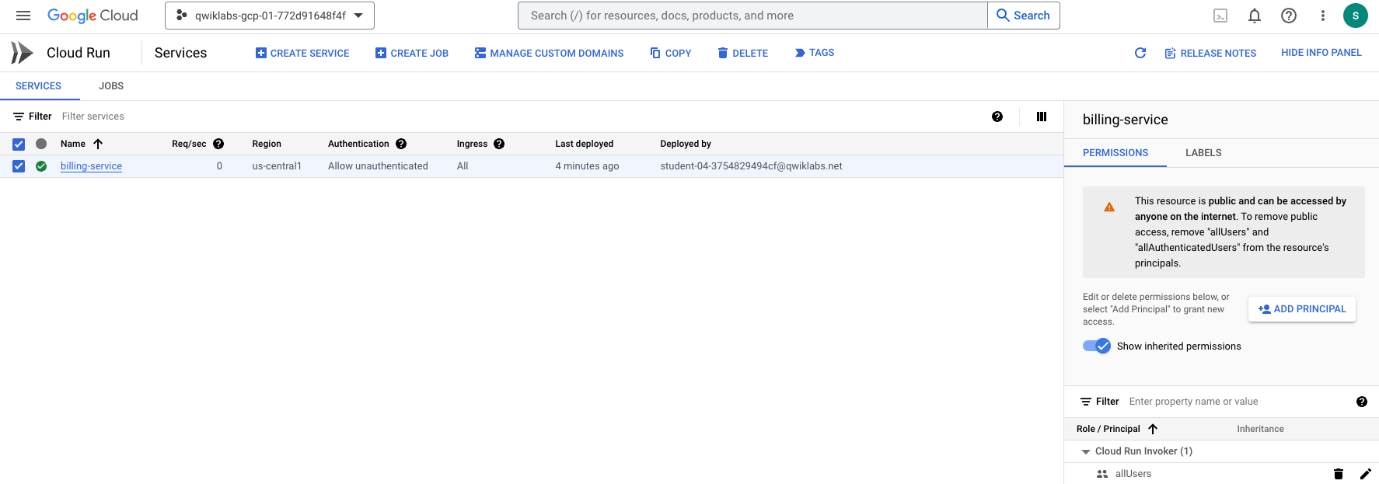
curl -X POST -H "Content-Type: application/json" $BILLING\_SERVICE\_URL -d '{"userid": "1234", "minBalance": 100}'

Copied!

content\_copy

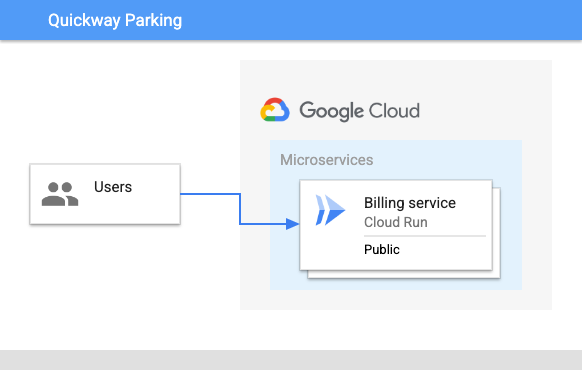
The service does not generate any output when invoked.

1. In the Google Cloud console **Navigation menu** (navmenu), click **Cloud Run**.
2. Click the link to the billing-service.
3. To view the service logs, click **Logs**.
4. Add the log filter minBalance to view the minimum balance received in the request made to the service.
5. To go back to the service details page, click **<- Service details**.
6. Select the billing-service by checking the box to the left of the green check mark.



The Security team has spotted something in the security settings. Can you see what part of the above configuration has them so concerned?

Take a closer look at the authentication applied. Currently **anyone on the internet** can call the billing service. This is indicated by the *allUsers* identity that has the *Cloud Run Invoker* role.



When the Billing service was originally deployed, it used the --allow-unauthenticated permission, which means that the service is publicly accessible, and can be invoked without any authentication.

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --allow-unauthenticated | Make the service publicly accessible (Unauthenticated users can access it). |
| Invoking Principal | allUsers | Allow the service to be invoked/triggered by anyone. |

By removing the --allow-unauthenticated permission you can use the Cloud Run default permissions to secure the service, or you can explicitly specify the no-allow-unauthenticated permission.

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --no-allow-unauthenticated | Secure the service with authentication (Only authenticated users can access it). |
| Invoking Principal | none | Do not allow the service to be invoked/triggered by anyone. |

**Note:** Remember, with Google Cloud, always try to use least privilege permissions in your solution.

By making these changes, the Security team will be a lot happier with the overall design.

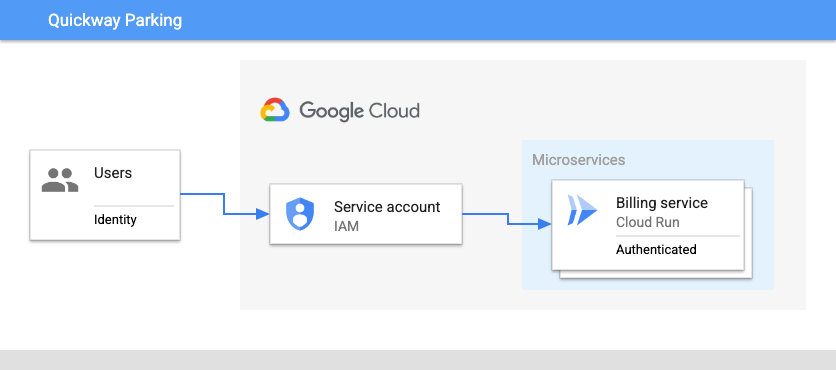
Click **Check my progress** to verify your performed task.

Deploy a public Cloud Run Service

Check my progress

## Task 3. Authenticating service requests

The team updates the application design to show how the changes will work:



The main changes are:

* Remove unauthenticated public access to the billing service.
* Create a new service account with appropriate permissions to invoke the billing service.

### Update the service to require authentication

Now that you understand more about the permissions used with Cloud Run, correct the authentication permissions applied to the Billing service:

1. Delete the existing deployed service:

gcloud run services delete billing-service

Copied!

content\_copy

1. If prompted, type **Y**, and then press **Enter**.
2. Redeploy the billing service with the --no-allow-authenticated permission:
3. gcloud run deploy billing-service \
4. --image gcr.io/qwiklabs-resources/gsp723-parking-service \
5. --region $LOCATION \

--no-allow-unauthenticated

Copied!

content\_copy

Redeploying the service means it no longer allows unauthenticated access at its service URL. In addition, the access permission to invoke the service has been removed.

1. Wait a few seconds, and then invoke the billing service again as before:

curl -X POST -H "Content-Type: application/json" $BILLING\_SERVICE\_URL -d '{"userid": "1234", "minBalance": 100}'

Copied!

content\_copy

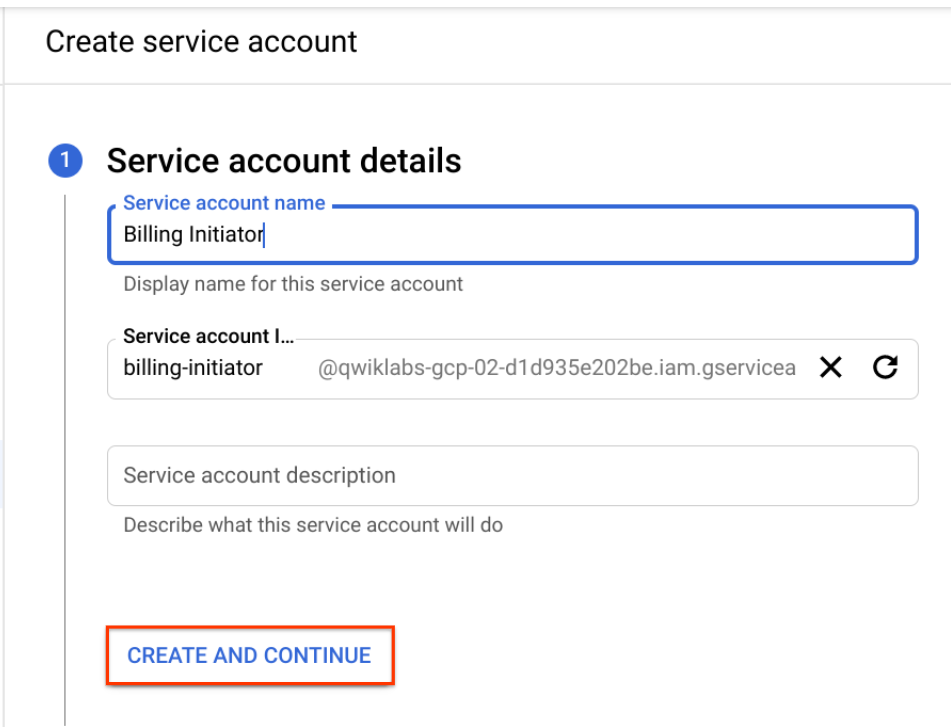
As expected, the output is a permissions error since the service now requires authentication.

### Create a service account

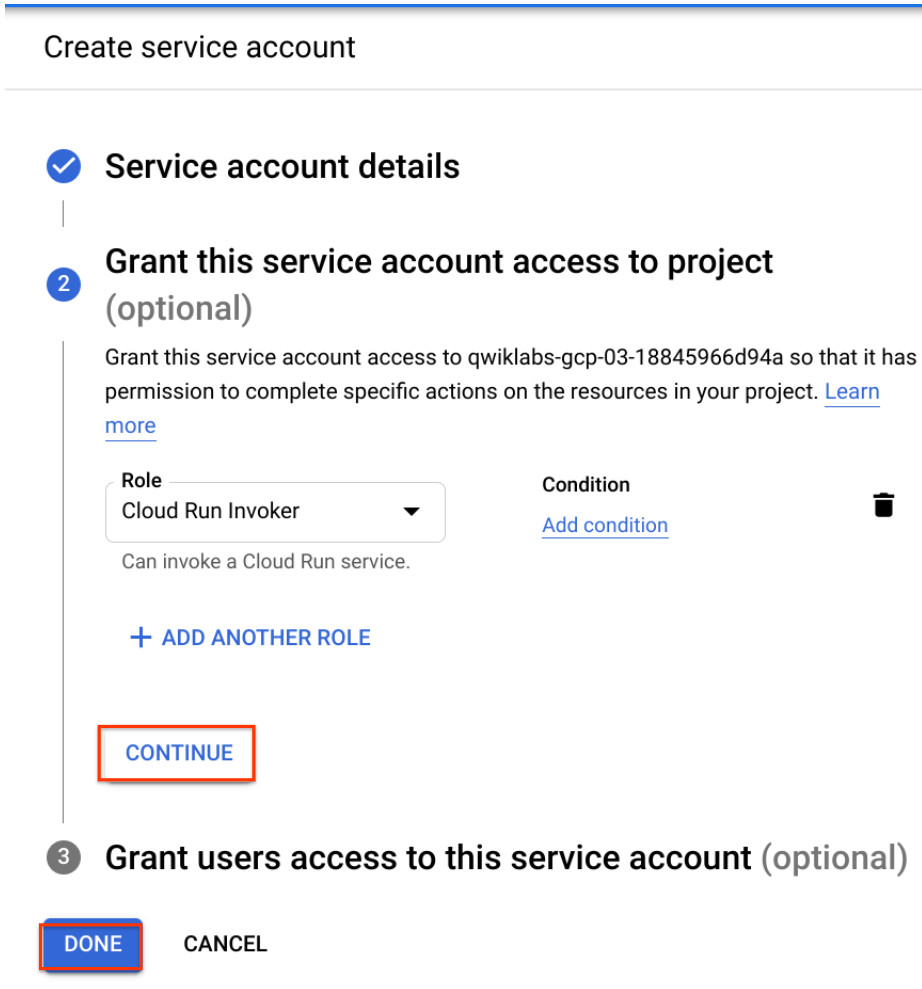
To invoke the billing service you will need an identity or service account with appropriate permissions, and bind that identity to the service.

This can be done in the Google Cloud console, or with the gcloud command line interface. In this lab, you use the Google Cloud console to create the service account and set up the new policy binding for the billing service.

1. In the Google Cloud console **Navigation menu** (navmenu), select **IAM & Admin > Service Accounts**.
2. To create a new service account that will provide authenticated access, click **Create Service Account** near the top.
3. Name the service account: Billing Initiator.



1. To create the account, click **Create and Continue**, and then advance to the **Grant Access** step.
2. To give the Billing Initiator service account permissions to invoke the billing service, select the **Role** drop-down, scroll the left side to Cloud Run, and then select the role Cloud Run Invoker.



1. To complete the setup of the service account, click **Continue**, and then click **Done**.

You will see the new service account at the top of the list of service accounts in the console.



The service account **Billing Initiator** has been created with the authorization to invoke a Cloud Run service, using an IAM policy binding on your project.

Click **Check my progress** to verify your performed task.

Create a service account

Check my progress

## Task 4. Invoke the service with authentication

Now that you have a service account with the appropriate permission, you can use it to invoke your Cloud Run service.

### Authenticate with gcloud

The first step is to set the service account in gcloud so it can be used to authenticate with the service.

1. In the Cloud Shell terminal menu, open a new shell in a separate tab by clicking **Add** (add).

Execute the remaining commands of this task in this Cloud Shell window.

1. Get the service account identity email and save it in an environment variable:

BILLING\_INITIATOR\_EMAIL=$(gcloud iam service-accounts list --filter="Billing Initiator" --format="value(EMAIL)"); echo $BILLING\_INITIATOR\_EMAIL

Copied!

content\_copy

1. In this Cloud Shell terminal, assign the URL of the billing service to an environment variable:
2. BILLING\_SERVICE\_URL=$(gcloud run services list \
3. --format='value(URL)' \

--filter="billing-service")

Copied!

content\_copy

1. To authenticate gcloud using the service account, generate a key file:

gcloud iam service-accounts keys create key.json --iam-account=${BILLING\_INITIATOR\_EMAIL}

Copied!

content\_copy

1. Authorize access to Cloud Run with a service account:

gcloud auth activate-service-account --key-file=key.json

Copied!

content\_copy

### Invoke the service

1. Invoke the Cloud Run billing service with an identity token generated from the service account:
2. curl -X POST -H "Content-Type: application/json" \
3. -H "Authorization: Bearer $(gcloud auth print-identity-token)" \

$BILLING\_SERVICE\_URL -d '{"userid": "1234", "minBalance": 500}'

Copied!

content\_copy

1. In the Google Cloud console **Navigation menu** (navmenu), click **Cloud Run**.
2. Click the link to the billing-service.
3. To view the service logs, click **Logs**.
4. Add the log filter minBalance to view the updated minimum balance received in the request made to the service.

Click **Check my progress** to verify your performed task.

Invoke a Cloud Run service with authentication

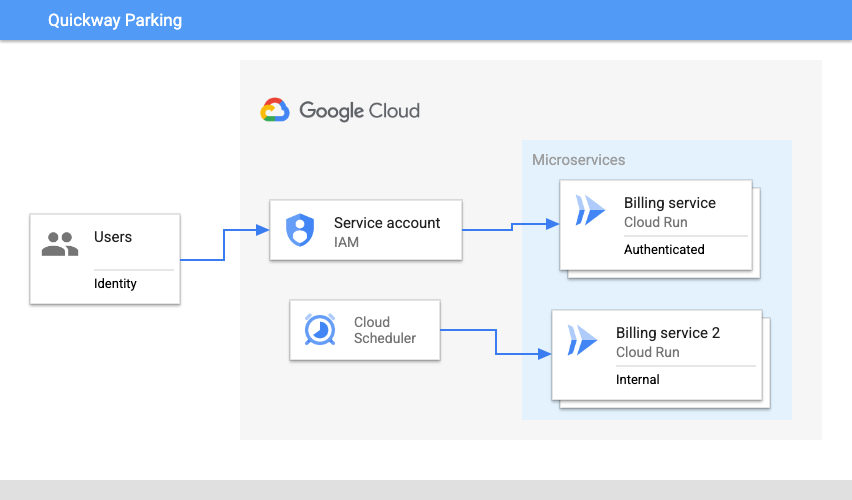
Check my progress

## Task 5. Implement least privilege

You've used a service account with the appropriate permissions to invoke a Cloud Run service that was previously accessible by anyone. But, have you used the absolute minimum privileges needed to call this specific service?

To determine if this is true, deploy a second billing service which we will assume should be accessible only by other internal private services, such as Cloud Scheduler.

Here's a diagram of this requirement:



### Deploy a second service

1. Open a third Cloud Shell terminal window or tab.
2. Create a LOCATION environment variable:

LOCATION=us-east1

Copied!

content\_copy

1. To simulate a second service, deploy the billing application image to Cloud Run:
2. gcloud run deploy billing-service-2 \
3. --image gcr.io/qwiklabs-resources/gsp723-parking-service \
4. --region $LOCATION \

--no-allow-unauthenticated

Copied!

content\_copy

1. Assign the URL of the new service to an environment variable:
2. BILLING\_SERVICE\_2\_URL=$(gcloud run services list \
3. --format='value(URL)' \

--filter="billing-service-2")

Copied!

content\_copy

### Invoke the second service with the service account identity

1. In this third Cloud Shell terminal, authorize access to Cloud Run with the same service account:

gcloud auth activate-service-account --key-file=key.json

Copied!

content\_copy

1. Invoke the second Cloud Run service with an identity token generated from the service account:
2. curl -X POST -H "Content-Type: application/json" \
3. -H "Authorization: Bearer $(gcloud auth print-identity-token)" \

$BILLING\_SERVICE\_2\_URL -d '{"userid": "1234", "minBalance": 900}'

Copied!

content\_copy

Why was this successful!? It's because when you created the service account, the Cloud Run Invoker permissions were granted to this account on the project. Because of inheritance, resources in the project such as the two Cloud Run services inherit those permissions, and as a result, the service account can be used to invoke the services.

### Restrict service account permissions

To fully implement least privilege, the service account should only be granted permissions on the service that it needs.

In this subtask, you remove the permission previously granted to the service account on the project, and then add the appropriate permissions required to invoke the original billing service.

1. Switch to the first Cloud Shell terminal window.
2. In this Cloud Shell terminal, get the service account identity email and save it in an environment variable:

BILLING\_INITIATOR\_EMAIL=$(gcloud iam service-accounts list --filter="Billing Initiator" --format="value(EMAIL)"); echo $BILLING\_INITIATOR\_EMAIL

Copied!

content\_copy

1. Remove the permission on the service account for the project:
2. gcloud projects remove-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT \
3. --member=serviceAccount:${BILLING\_INITIATOR\_EMAIL} \

--role=roles/run.invoker

Copied!

content\_copy

1. Add the permission to the service account on the billing service:
2. gcloud run services add-iam-policy-binding billing-service --region $LOCATION \
3. --member=serviceAccount:${BILLING\_INITIATOR\_EMAIL} \

--role=roles/run.invoker --platform managed

Copied!

content\_copy

### Invoke the services

1. Switch back to the second Cloud Shell terminal window or tab.
2. Wait a few seconds, and then invoke the first Cloud Run billing service with an identity token generated from the service account:
3. curl -X POST -H "Content-Type: application/json" \
4. -H "Authorization: Bearer $(gcloud auth print-identity-token)" \

$BILLING\_SERVICE\_URL -d '{"userid": "1234", "minBalance": 700}'

Copied!

content\_copy

It takes a few seconds for the updated permissions to propagate, after which this invocation should be successful.

1. Switch to the third Cloud Shell terminal window.
2. Try to invoke the second Cloud Run service with an identity token generated from the same service account:
3. curl -X POST -H "Content-Type: application/json" \
4. -H "Authorization: Bearer $(gcloud auth print-identity-token)" \

$BILLING\_SERVICE\_2\_URL -d '{"userid": "1234", "minBalance": 500}'

Copied!

content\_copy

You should now receive a permissions error indicating that the service account has only the minimum set of permissions required to invoke the first service.

Click **Check my progress** to verify your performed task.

Use least privilege to invoke a Cloud Run service

Check my progress

## Congratulations!

In this lab, you have seen how to reconfigure a deployed service to secure access to it, and implemented the principle of least privilege when granting permissions to access resources on Google Cloud. You:

* Deployed a service to Cloud Run.
* Used the gcloud CLI to update the service to require authentication.
* Created a service account with the required permissions to invoke the service.
* Set the minimum permissions required to invoke a specific service on Cloud Run, implementing least privilege.

Follow the [Serverless Expeditions video series](https://www.youtube.com/playlist?list=PLIivdWyY5sqJwq_pgOxcHzusWjXDVCEiX) to learn more about how to use these products within your project.

## 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 2022 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.

### [Cloud Code and Duet AI IDE Plugins](https://cloud.google.com/code)

# Using Cloud PubSub with Cloud Run [APPRUN]

45 minutes1 Credit

## Overview

Pub/Sub enables applications to take advantage of efficient message queues. The service is compatible with a range of Google Cloud services, and in this lab, you learn how to integrate it with Cloud Run.

This lab is based on resolving a customer use case by using serverless infrastructure. The lab features three high level sections that resolve a technical problem:

* Situational Overview
* Requirements Gathering
* Developing a minimal viable product

### Objectives

In this lab, you learn to:

* Enable the Cloud Run API.
* Deploy microservices to Cloud Run.
* Create a Pub/Sub topic.
* Invoke a Cloud Run service from a Pub/Sub subscription.

### Prerequisites

These labs are based on intermediate knowledge of Google Cloud. While the steps required are covered in the content, it would be helpful to have familiarity with any of the following products:

* Pub/Sub
* Cloud Run

## Setup and requirements

#### Before you click the Start Lab button

**Note: Read these instructions.**

Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

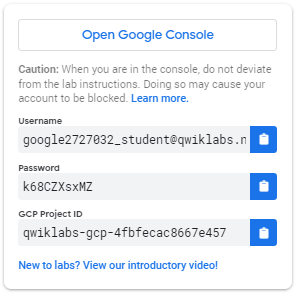
To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:**If you already have your own personal Google Cloud account or project, do not use it for this lab.**Note:**If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Console

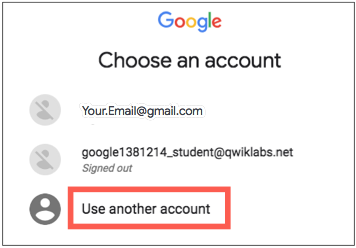
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

**Note:**Open the tabs in separate windows, side-by-side.

1. On the Choose an account page, click **Use Another Account**. The Sign in page opens.



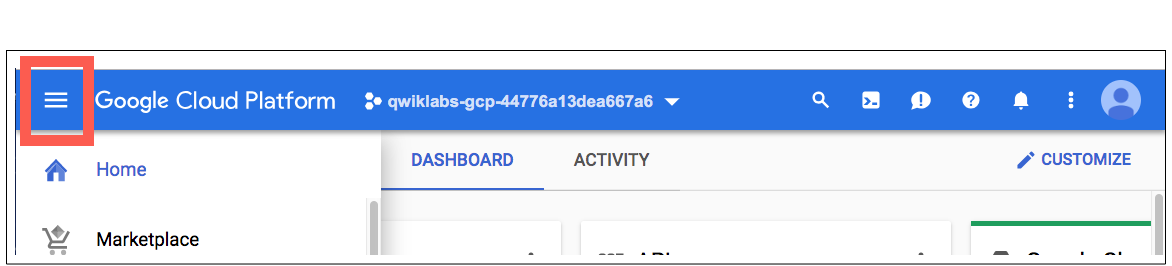
1. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

**Note:**You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

1. Click through the subsequent pages:

* Accept the terms and conditions.
* Do not add recovery options or two-factor authentication (because this is a temporary account).
* Do not sign up for free trials.

After a few moments, the Cloud console opens in this tab.

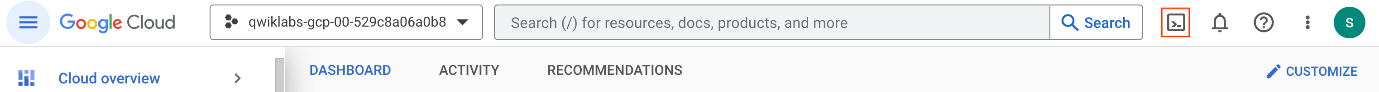
**Note:**You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

### Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

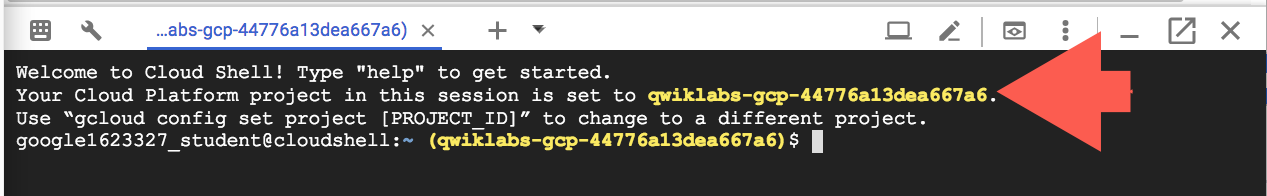
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



**gcloud** is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Situational overview



In this lab, you will help the development team at Critter Junction investigate the use of Pub/Sub for their requirements. The team would like to explore how to perform efficient queue processing within their applications.

### Requirements gathering

The team at Critter Junction has a public web application and several microservices built on Google Cloud. Communication between the microservices is critical and needs a resilient form of messaging to be established between each application component.

The development team's previous attempts were unsuccessful due to the microservices needing to know a lot about each other ( [High Coupling](https://en.wikipedia.org/wiki/Coupling_(computer_programming))). In addition, if a service was temporarily unavailable, messages would be lost.

The team needs a solution that includes a level of resilience without introducing additional service dependencies (Low Coupling) into their systems. Now that you know a bit more about Critter Junction and the issues they face, try to prioritize the key criteria for a solution.

### Defining Critter Junction priorities

To ascertain the key use cases and priorities, initial discussions are held with the Critter Junction stakeholders. The results of the discussions are shown below:

|  |  |
| --- | --- |
| **Ref** | **User Story** |
| 1 | As a lead developer, I want to ensure that messaging is resilient, so service operations will be restored without needing manual intervention. |
| 2 | As a program manager, I want services to be capable of scaling seamlessly so additional transactional load does not lead to system instability. |
| 3 | As an operations lead, I want services to be managed so staff does not need to be reassigned from important maintenance work. |

From a discussion with the team leads, the following high level tasks are defined:

|  |  |
| --- | --- |
| **Ref** | **Definition of Done** |
| 1 | Establish an asynchronous component for inter-service communication. |
| 2 | Implement the proven scalability of the solution. |
| 3 | Services must run unsupervised. |

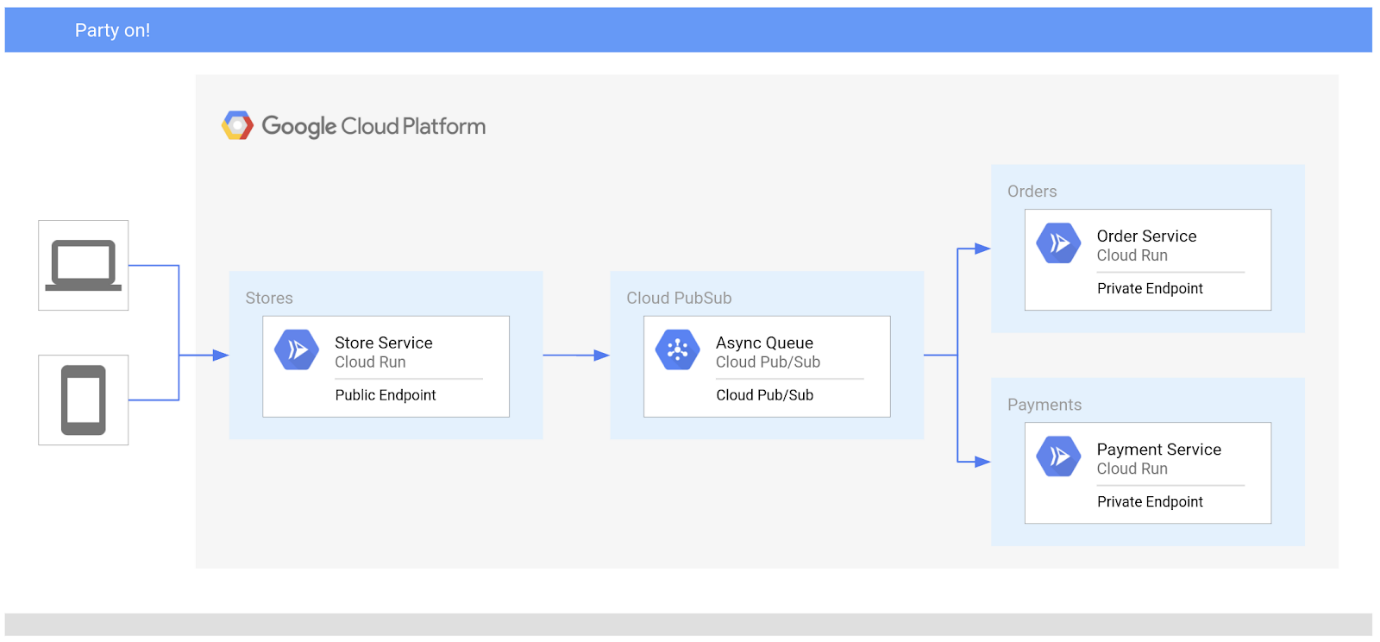
The team at Critter Junction is keen to define a solution that can be implemented quickly. In consideration of the requirements, the development team narrows their options down to:

* Pub/Sub
* Cloud Tasks

See [Pub/Sub versus Cloud Tasks](https://cloud.google.com/pubsub/docs/tasks-vs-pubsub) to learn more.

|  |  |  |
| --- | --- | --- |
| **Product** | **Use case** | **Choice** |
| **Pub/Sub** | **"Optimal for more general event data ingestion and distribution patterns where some degree of control over execution can be sacrificed."** | green check mark |
| Cloud Tasks | "Appropriate for use cases where a task producer needs to defer or control the execution timing of a specific webhook or remote procedure call." | red incorrect mark |

After considering the requirements, the development team chooses Pub/Sub because they only require a push based distribution pattern. The following high level architecture diagram summarizes the minimal viable product (MVP) that they need to investigate.

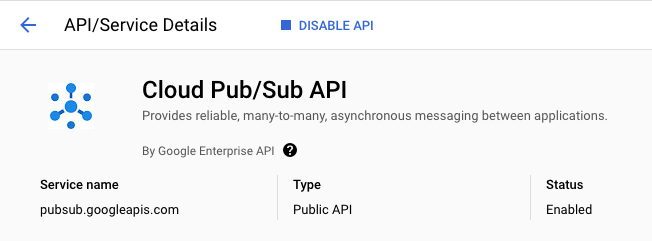


In the proposed solution, Pub/Sub will be used to handle asynchronous messages between services.

## Task 1. Ensure that the Pub/Sub API is successfully enabled

To ensure access to the necessary API, re-enable the **Pub/Sub** API.

1. In the Google Cloud console **Navigation menu** (navmenu), under **APIs & Services**, click **Library**.
2. In the **Search** box, type **Pub/Sub**
3. Click the result for **Cloud Pub/Sub API**.
4. Click **Manage**.
5. Click **Disable API**. If asked to confirm, click **Disable**.
6. Again, when prompted Do you want to disable Cloud Pub/Sub API and its dependent APIs?, Click **Confirm**.
7. To re-enable the API, click **Enable**.

When the API has been re-enabled, the page displays information about the API. 

## Task 2. Developing a minimal viable product (MVP)

Critter Junction has multiple Cloud Run services that they would like integrated with Pub/Sub. To build an MVP, the following tasks are required:

* Deploy a producer service
* Deploy a consumer service
* Create a service account
* Create a Pub/Sub topic

### Deploy a producer service

Critter Junction specifies that the externally facing store service should be configured as a public endpoint, indicating these requirements:

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --allow-unauthenticated | Make the service PUBLIC (Unauthenticated users can see it). |
| Invoke Permission | allUsers | Allow the service be invoked/triggered by anyone. |

The producer store service accepts public internet based connections for purchase orders. To do this, the service must not require authentication and must be able to be triggered by anyone.

Information collected by this service will be passed to the backend consumer services.

Configure and deploy the store service on Cloud Run. Execute the following commands in Cloud Shell.

1. Enable the **Cloud Run** API and configure your Shell environment:

gcloud services enable run.googleapis.com

Copied!

content\_copy

1. Create a LOCATION environment variable:

LOCATION=us-central1

Copied!

content\_copy

1. Set the compute region:

gcloud config set compute/region $LOCATION

Copied!

content\_copy

1. Deploy the store service:

gcloud run deploy store-service \

--image gcr.io/qwiklabs-resources/gsp724-store-service \

--region $LOCATION \

--allow-unauthenticated

Copied!

content\_copy

Click Check my progress to verify the objective.

Deploy the Cloud Run store service

Check my progress

Once the store service is deployed, the store service is publicly accessible over the internet.

### Deploy a consumer service

The development team also needs to configure the order service that can be accessed at a private endpoint. Unlike the store service, the order service is not meant to be publicly accessible over the internet, and should only be invoked by an account with the appropriate permissions.

For Cloud Run based services, this can be achieved by using the following settings:

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --no-allow-unauthenticated | Make the service PRIVATE (Only authenticated users can see it). |
| Invoke Role/Permission | Cloud Run Invoker | Only allow the service to be invoked by an account with the Cloud Run Invoker role. |

Configure and deploy the order service:

gcloud run deploy order-service \

--image gcr.io/qwiklabs-resources/gsp724-order-service \

--region $LOCATION \

--no-allow-unauthenticated

Copied!

content\_copy

Click Check my progress to verify the objective.

Deploy the Cloud Run order service

Check my progress

Now only authenticated accounts can access and invoke the service.

## Pub/Sub overview

Pub/Sub is an asynchronous messaging service that decouples services that produce events from services that consume and process events.

Pub/Sub core concepts:

* Topic
* Subscription
* Message
* Message attribute

Pub/Sub requires a couple of options to be completed prior to successful deployment. In the Google Cloud console, Pub/Sub can be accessed under the Big Data menu option.

|  |  |
| --- | --- |
| **Field** | **Description** |
| Topic | A named resource to which messages are sent by publishers. |
| Subscription | A named resource representing the stream of messages from a single, specific topic, to be delivered to the subscribing application. For more details about subscriptions and message delivery semantics, see the [Subscriber Guide](https://cloud.google.com/pubsub/subscriber). |
| Message | The combination of data and (optional) attributes that a publisher sends to a topic and is eventually delivered to subscribers. |
| Message attribute | A key-value pair that a publisher can define for a message. For example, key **iana.org/language\_tag** and value **en** could be added to messages to mark them as readable by an English-speaking subscriber. |

Pub/Sub can be used in a wide variety of use cases, the most common of which are listed below:

|  |  |
| --- | --- |
| **Use Case** | **Example** |
| Balancing workloads in network clusters | For example, a large queue of tasks can be efficiently distributed among multiple workers, such as Compute Engine instances. |
| Implementing asynchronous workflows | For example, an order processing application can place an order on a topic, from which it can be processed by one or more workers. |
| Distributing event notifications | For example, a service that accepts user signups can send notifications whenever a new user registers, and downstream services can subscribe to receive notifications of the event. |
| Refreshing distributed caches | For example, an application can publish invalidation events to update the IDs of objects that have changed. |
| Logging to multiple systems | For example, a Google Compute Engine instance can write logs to the monitoring system, to a database for later querying, and so on. |
| Data streaming from various processes or devices | For example, a residential sensor can stream data to backend servers hosted in the cloud. |
| Reliability improvement | For example, a single-zone Compute Engine service can operate in additional zones by subscribing to a common topic, to recover from failures in a zone or region. |

## Task 3. Deploying Pub/Sub

Now that the producer (store service) and consumer (order service) services have been successfully deployed, you can focus on the main features of Pub/Sub. Using Pub/Sub requires two activities:

* Create a Topic
* Create a Subscription

### Create a Topic

When an asynchronous (push) event is created on a topic, applications that subscribe to the topic will be able to process the associated messages. [Push event processing with Pub/Sub](http://cloud.google.com/run/docs/events/pubsub-push) provides a scalable way to handle messaging on Google Cloud.

The new Pub/Sub Topic will have following values.

|  |  |
| --- | --- |
| **Field** | **Value** |
| Name | ORDER\_PLACED |
| Encryption | Google-managed key |

1. Create a Topic in Pub/Sub:

gcloud pubsub topics create ORDER\_PLACED

Copied!

content\_copy

**Note:**Messages that are sent using Pub/Sub are encoded as base64 on transmission, and need to be decoded on receipt.

Click Check my progress to verify the objective.

Create a Pub/Sub Topic

Check my progress

By creating the Pub/Sub Topic, messages can now be independently stored and delivered in a resilient manner.

You'll create a subscription in a subsequent task.

## Task 4. Creating a service account

To deliver a Pub/Sub message to a Cloud Run service, you need a Pub/Sub subscription. The subscription must be able to invoke the service using a service account with the appropriate permissions. In this lab, the consumer order service will be invoked by a subscription using the service account.

To achieve this functionality, the following activities are required:

* Create a Service Account
* Bind the Invoker Role permissions to the service account

### Service account creation

Create a new service account that will provide authenticated access.

1. Create a new service account called **Order Initiator**:
2. gcloud iam service-accounts create pubsub-cloud-run-invoker \

--display-name "Order Initiator"

Copied!

content\_copy

1. Confirm that the service account has been created:

gcloud iam service-accounts list --filter="Order Initiator"

Copied!

content\_copy

Click Check my progress to verify the objective.

Create a Service Account

Check my progress

At this point, the **Order Initiator** service account is available. However, it does not have a role or permissions assigned. To assign it IAM permissions, you need to apply or bind role permissions to the service account.

### Bind role permissions

To bind permissions to an account that is used to invoke a service on Cloud Run, you need the following information:

|  |  |
| --- | --- |
| **Category** | **Description** |
| Service Name | The name of the deployed service to be invoked. |
| Member | The account to bestow the role permissions. |
| Region | The region in which the service is deployed. |
| Platform | The platform type (Cloud Run Managed, Cloud Run for Anthos, or Cloud Run for VMWare) |

1. Bind the service account with the role Cloud Run Invoker on the order service:
2. gcloud run services add-iam-policy-binding order-service --region $LOCATION \
3. --member=serviceAccount:pubsub-cloud-run-invoker@$GOOGLE\_CLOUD\_PROJECT.iam.gserviceaccount.com \

--role=roles/run.invoker --platform managed

Copied!

content\_copy

The new service account has now been given permissions to invoke a Cloud Run service.

1. Create an environment variable to store the project number:
2. PROJECT\_NUMBER=$(gcloud projects list \
3. --filter="qwiklabs-gcp" \

--format='value(PROJECT\_NUMBER)')

Copied!

content\_copy

1. Enable the project service account to create tokens:
2. gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT \
3. --member=serviceAccount:service-$PROJECT\_NUMBER@gcp-sa-pubsub.iam.gserviceaccount.com \

--role=roles/iam.serviceAccountTokenCreator

Copied!

content\_copy

## Task 5. Create a Pub/Sub subscription

In this task, you create the Pub/Sub subscription and configure it to use the new service account.

1. Create an environment variable to store the endpoint of the order service:
2. ORDER\_SERVICE\_URL=$(gcloud run services describe order-service \
3. --region $LOCATION \

--format="value(status.address.url)")

Copied!

content\_copy

1. Create a subscription and bind it to the order service:
2. gcloud pubsub subscriptions create order-service-sub \
3. --topic ORDER\_PLACED \
4. --push-endpoint=$ORDER\_SERVICE\_URL \

--push-auth-service-account=pubsub-cloud-run-invoker@$GOOGLE\_CLOUD\_PROJECT.iam.gserviceaccount.com

Copied!

content\_copy

Click Check my progress to verify the objective.

Create a subscription

Check my progress

## Task 6. Testing the application

To test the application, send a sample JSON payload to the store service.

1. Create a file called test.json with the following content. You can use your choice of editor such as nano, vi, or the Cloud Shell editor.
2. {
3. "billing\_address": {
4. "name": "Kylie Scull",
5. "address": "6471 Front Street",
6. "city": "Mountain View",
7. "state\_province": "CA",
8. "postal\_code": "94043",
9. "country": "US"
10. },
11. "shipping\_address": {
12. "name": "Kylie Scull",
13. "address": "9902 Cambridge Grove",
14. "city": "Martinville",
15. "state\_province": "BC",
16. "postal\_code": "V1A",
17. "country": "Canada"
18. },
19. "items": [
20. {
21. "id": "RW134",
22. "quantity": 1,
23. "sub-total": 12.95
24. },
25. {
26. "id": "IB541",
27. "quantity": 2,
28. "sub-total": 24.5
29. }
30. ]

}

Copied!

content\_copy

1. Create an environment variable to store the endpoint of the store service:
2. STORE\_SERVICE\_URL=$(gcloud run services describe store-service \
3. --region $LOCATION \

--format="value(status.address.url)")

Copied!

content\_copy

1. To test communication between the microservices and generate an order ID, post a message to the store service:

curl -X POST -H "Content-Type: application/json" -d @test.json $STORE\_SERVICE\_URL

Copied!

content\_copy

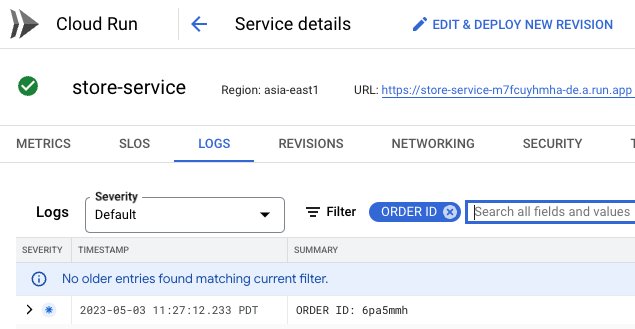
The output of the command indicates that an order had been successfully created, and should be similar to:

{"status":"success","order\_id":"6pa5mmh"}

### Store service

The store service (public endpoint) uses Pub/Sub to transmit information to the order service (private endpoint).

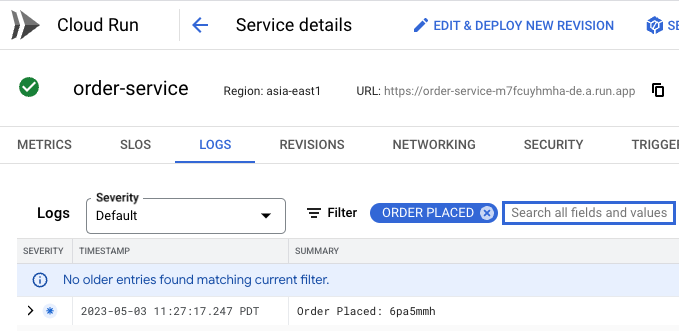
1. In the Google Cloud console **Navigation menu** (navmenu), click **Cloud Run**.
2. Click the link to the store-service.
3. To view the service logs, click **Logs**. Check the store service logs to view the order ID that was generated.
4. Add the log filter ORDER ID to see the ID generated by the store service.



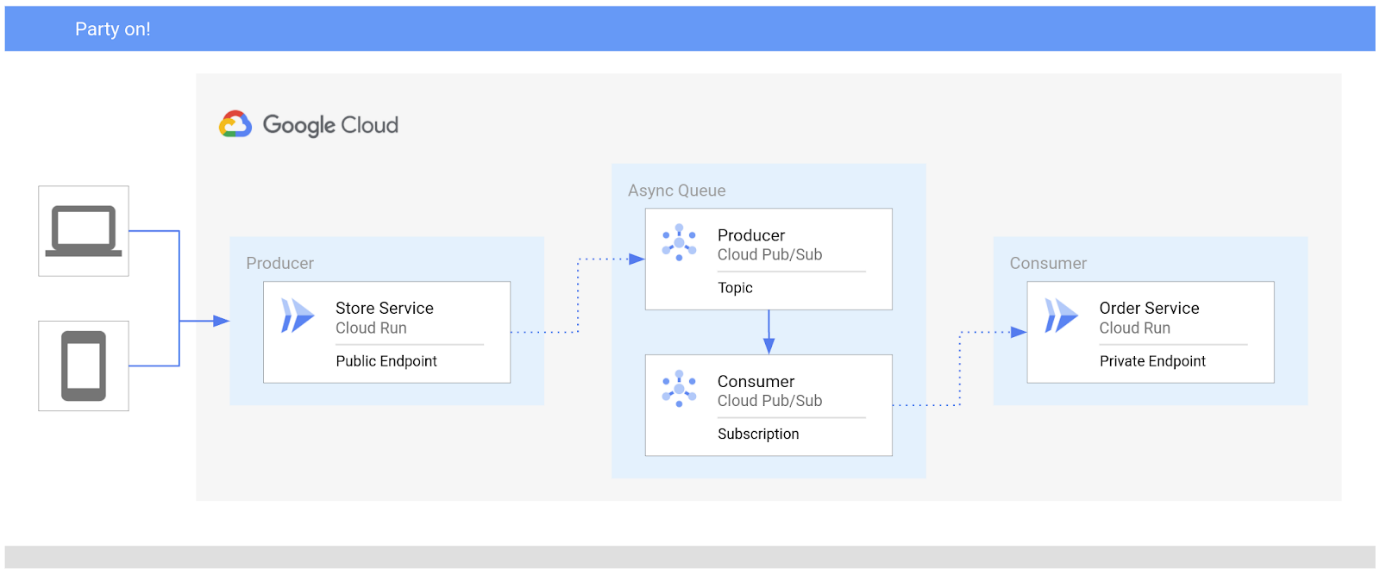
### Order service

The order service receives a message from the store service passed with Pub/Sub.

1. Check the order service logs to confirm that the JSON data was successfully transferred.
2. Add the log filter Order Placed to see the generated order ID that was passed to the order service.



Critter Junction have now updated their solution to take advantage of Pub/Sub. The following high level architecture diagram summaries the solution deployed.



You have successfully deployed Pub/Sub on Google Cloud to asynchronously communicate between Cloud Run services.

## Congratulations!

In this lab, you learned how to integrate Cloud Run services with Pub/Sub in your Google Cloud infrastructure. You learned how to:

* Deploy services to Cloud Run
* Create a Service Account with the appropriate role and permissions
* Define a Pub/Sub Topic
* Bind a Pub/Sub Subscription to a Service Account

### Next steps / Learn more

Follow the [Serverless Expeditions video series](https://www.youtube.com/watch?v=s2TIWIzCftM&list=PLIivdWyY5sqJwq_pgOxcHzusWjXDVCEiX) to learn more about how to use these products within your project.

* [Cloud Run](https://cloud.google.com/run)
* [Cloud Tasks](https://cloud.google.com/tasks)
* [Cloud Functions](https://cloud.google.com/functions)

**Manual Last Updated May 9, 2023**

**Lab Last Tested May 9, 2023**

Copyright 2022 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.