O'REILLY®

Google Cloud Cookbook

Practical Solutions for Building and Deploying Cloud Services

Rui Costa & Drew Hodun

O'REILLY®

Google Cloud Cookbook

Get quick hands-on experience with Google Cloud. This cookbook provides a variety of self-contained recipes that show you how to use Google Cloud services for your enterprise application. Whether you're looking for practical ways to apply microservices, AI, analytics, security, or networking solutions, these recipes take you step-by-step through the process and provide discussions that explain how and why the recipes work.

Ideal for system engineers and administrators, developers, network and database administrators, and data analysts, this cookbook helps you get started with Google Cloud regardless of your level of experience. Google veterans Rui Costa and Drew Hodun also cover advanced-level Google Cloud services for those who have appreciable experience with the platform.

- Learn how to get started with Google Cloud
- Understand the depth of services Google Cloud provides
- Gain hands-on experience using practical examples and labs
- Explore topics that include BigQuery, Cloud Run, and Kubernetes
- Build and run mobile and web applications on Google Cloud
- Examine ways to build your cloud applications for scale
- Build a minimum viable product (MVP) app to use in production
- Learn data platform and pipeline skills

"Tired of diving into GCP and finding only "Hello World" examples? *Google Cloud Cookbook* contains recipes for real-world applications, from serverless to security, including microservices, big data, AI, ML, Firebase, security, networking, and Kubernetes applications."

—Aurélie Vache
DevRel at OVHcloud, Google Developer Expert on Google Cloud, CNCF Ambassador, and Docker captain

Rui Costa has worked at Google in several roles, most recently as a learning consultant working with strategic customers and partners on their Google Cloud learning plans.

Drew Hodun has had a number of roles at Google, all customer-facing and focused on machine learning applications running on Google Cloud.

CLOUD

US $59.99 CAN $79.99
ISBN: 978-1-492-09289-6

Google Cloud Cookbook

Practical Solutions for Building and Deploying Cloud Services

Rui Costa and Drew Hodun

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Google Cloud Cookbook
by Rui Costa and Drew Hodun

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Jennifer Pollock
Development Editor: Jeff Bleiel
Production Editor: Caitlin Ghegan
Copyeditor: nSight, Inc.
Proofreader: Kim Wimpsett
Indexer: nSight, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

October 2021: First Edition

Revision History for the First Edition
2021-10-07: First Release

See *https://oreil.ly/ko0Ge* for release details.

978-1-492-09289-6

[LSI]

Table of Contents

Preface

Building applications has never been easier or more exciting than in the era of the public cloud. Engineers can build faster, better, and bigger with the myriad of cloud tools and providers available to them. As organizations continue adopting the cloud to run their workloads, engineers need to learn new skills continually to run these workloads successfully in the cloud. This book provides you with recipes that we feel will help you get started running your workloads on Google Cloud.

Who Should Read This Book

If you are a software engineer, cloud architect, data scientist, site reliability engineer, or sysadmin, and you're working with Google Cloud, this book is for you. As an engineer working with Google Cloud services, you need to understand all the services available to you, including how to leverage them in your code, scripts, and solutions. If you are new to the cloud or moving from another cloud provider, the recipes and code samples within will quickly familiarize you with building on Google Cloud. More experienced Google Cloud engineers will still benefit from some of the more advanced techniques at the end of every chapter or from chapters covering an area less familiar to them. These recipes were created and selected from years of experience getting Google Cloud customers up and running quickly.

Why I Wrote This Book

Rui Costa

Before joining Google, I worked as a consultant where I helped organizations adopt cloud services. As a consultant, I worked with organizations to understand their business and technical requirements. I helped them build architectures and deployment strategies to migrate or build their applications successfully in the cloud.

In my first three years at Google, I worked in a similar role, with a strong emphasis on Google Cloud. About two years ago, I took a new role at Google as a learning consultant. In my current role, I build custom learning content for Google Cloud. Building new learning programs requires me to stay updated as much as the engineers who will be taking the course and using it to deploy their applications on Google Cloud. For this, I always have to stay up-to-date with the services Google Cloud provides.

When building these courses, I'm always referring to the Google Cloud documentation website, collaborating with subject matter experts, and learning from my students. This process takes time, and I yearned for a book that I could reference for quick, repeatable recipes. I decided to write this book as a reference for all to use and gain quick access to recipes that you can use in your journey with Google Cloud.

I also have a strong background in software engineering and tried to apply this experience to the recipes in this book. You will find recipes that are not just applicable to using Google Cloud services but also to how to build your application with Google Cloud if it's based on Java, Go, Python, or Node.js. There is something for everyone.

Drew Hodun

Working for years as a customer-facing engineer at Google, I helped customer after customer and engineer after engineer get started on Google Cloud. I still love those aha moments I witness when an engineer opens Cloud Shell in-browser for the first time or runs their first BigQuery query that processes 100 billion regex expressions in 20 seconds. They are excited because they see the possibility to build better, bigger, faster.

With this book, I hope to share many of the tips, tricks, and getting-started knowledge that we've built up over the years onboarding customers to Google Cloud and wish I'd known at the beginning of my cloud journey. Although this book covers the products and services of Google Cloud as well as some common architectural patterns, it also contains slick shortcuts and advice throughout to make you a more experienced Google Cloud engineer.

Navigating This Book

This book is organized roughly as follows:

Chapter 1, "Introduction"

In this chapter, we cover all the fundamental concepts of Google Cloud, from projects and billing accounts to storage buckets. We'll also introduce you to many of the tools you'll use day to day.

Chapter 2, "Cloud Functions"

In this chapter, we present a range of recipes, from introductory recipes you can use to send emails or respond to SMS messages to advanced recipes that show you how to integrate CI/CD into your development workflow and integrate with Cloud Endpoints for API management.

Chapter 3, "Google Cloud Run"

In this chapter, you will learn how to trigger a Cloud Run service from a Pub/Sub topic creating automated pipelines, use your custom domain to increase your branding efforts by using a custom domain, and run blue/green deployments to release your application by directing traffic between two Cloud Run services running different versions of an application.

Chapter 4, "Google App Engine"

In this chapter, you will learn how to deploy your application with a CI/CD pipeline, secure it, map a custom domain, use ML APIs, and debug your application.

Chapter 5, "Google Cloud Compute Engine"

This chapter contains recipes for creating and managing your virtual machines. You'll find recipes for unique methods to automate deployments, deploy containers to virtual machines, and use Identity-Aware Proxy (IAP) to tunnel RDP traffic to connect to your Windows virtual machines securely.

Chapter 6, "Google Cloud Kubernetes Engine"

This chapter contains recipes for creating and managing your containers, including methods to automate deployments and deploy real-world applications by using MongoDB and Java applications.

Chapter 7, "Working with Data"

This chapter contains recipes for working with common data storage systems, including Cloud Storage, persistent disks on VMs, and databases like Cloud Spanner and Firestore.

Chapter 8, "BigQuery and Data Warehousing"

In this chapter, you will perform basic tasks in BigQuery as well as learn more about clustering, partitioning, undeleting data, and improving query performance.

Chapter 9, "Data Processing Tools"

This chapter covers some of the scalable data processing platforms, including Dataproc, Dataflow, and Data Fusion. You'll run your first pipelines on each of these platforms and learn some more advanced techniques on Dataflow.

Chapter 10, "AI/ML"

This chapter will get you up and running with Google's AI technologies. You'll create a hosted Jupyter Notebook and get started training TensorFlow ML models on the cloud and serving them on the cloud.

Chapter 11, "Google Cloud Security and Access"

In this chapter, you will learn how to create a service account to allow applications to access Google Cloud resources securely, implement authentication for applications running on Google Kubernetes Engine (GKE), run asset reports, and build a deny and allow list for your applications.

Chapter 12, "Google Cloud Networking"

This chapter covers concepts that users require to get started with Google Cloud networking, including securing your virtual machines, automating deployments of networking resources, and protecting your projects from data exfiltration.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a general note.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/ruiscosta/oreilly-google-cloud-cookbook*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Google Cloud Cookbook* by Rui Santos Costa and Andrew Hodun (O'Reilly). Copyright 2022 Drew Hodun and Rui Costa, 978-1-492-09289-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

> O'Reilly Media, Inc.
> 1005 Gravenstein Highway North
> Sebastopol, CA 95472
> 800-998-9938 (in the United States or Canada)
> 707-829-0515 (international or local)
> 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/GCCookbook*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *http://oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://youtube.com/oreillymedia*

Acknowledgments

We would like to thank Hussein Sharif and James Duncan for contributing several recipes to this book.

Thanks to Iain Foulds, Michael Hopkins and Aurélie Vache for their insightful and thorough technical reviews.

And of course, many thanks to our team at O'Reilly—to Jennifer Pollock for bringing us on board and proposing the book, to Jeff Bleiel for accompanying us along the many months of writing, to Caitlin Ghegan for the last mile burst of enthusiasm, and to Kerin Forsyth for improving so greatly upon the content.

Rui Costa

With all my love, I want to thank my family for their support and for always pushing me to grow: my wife Isabel, son Filipe, parents Germano and Maria, sister Sonia, brother-in-law Michael, nephew Thomas, father-in-law Joao, and mother-in-law Maria.

I also want to thank Anthony Okwechime, Lauren Kapnick, Russell Goldenbroit, and Casey Palowitch for always being there for me and believing in me.

Drew Hodun

A huge thanks to all who have helped me in my personal and professional journey, who have taught me to learn and grow joyously and to remember to be humble.

Thanks to Bob Scarborough, Italo Brito, and Lukman Ramsey for believing in me and pushing me to the next level.

To Dad for his curiosity. To Mom for her wisdom. To Rose for her spirit.

CHAPTER 1

Introduction

Welcome to Google Cloud! Whether you're a seasoned cloud developer or a sysadmin moving to the cloud for the first time, you'll find Google Cloud a great platform for building everything from the smallest serverless apps to the largest enterprise applications and data pipelines.

This chapter is a high-level overview of the platform, with some tips on how to get started. If you are already using Google Cloud, you can likely skip this chapter. If you are new to the platform, whether coming from on-premises (on-prem) or another cloud provider, you'll find here a brief overview of the services, tools, and features you need to know to get started. Toward the end of the chapter, we'll describe how to translate your existing cloud provider knowledge quickly. Future chapters are all traditional collections of recipes you'll find in a cookbook.

Overview

Like most cloud platforms, Google Cloud is a collection of physical resources such as servers and storage as well as higher-level services like BigQuery or AppEngine built on top of them, all running in Google data centers. The platform is organized into zones, regions, multiregions (continents), and then the entire globe. Table 1-1 shows a rough mental model of what each of these means.

Table 1-1. Service regionality description

Service regionality	Description
Zonal	A large cluster of compute in one or several data center buildings. Should be considered a single failure domain.[a]
Regional	A cluster of data center buildings (campus). Made of multiple zones. Each zone is generally on a different power and network infrastructure and on a different upgrade schedule. Regional services will be served out of multiple zones or automatically fail over in the case of a zonal outage.

Service regionality	Description
Multiregional	A group of regions generally on the same continent. Multiregional services will be served out of multiple regions and handle a regional failure.
Global	A service deployed across Google Cloud regions around the globe.

[a] Technically, a zone is a logical abstraction of a region with the failure domain properties described, though this physical analogy can be helpful to understand the difference and is often how they are deployed.

Different services have different scopes; for example, a virtual machine (VM) lives in a zone, whereas an L7 network load balancer exists globally. Figure 1-1 shows an example of some basic compute resources.

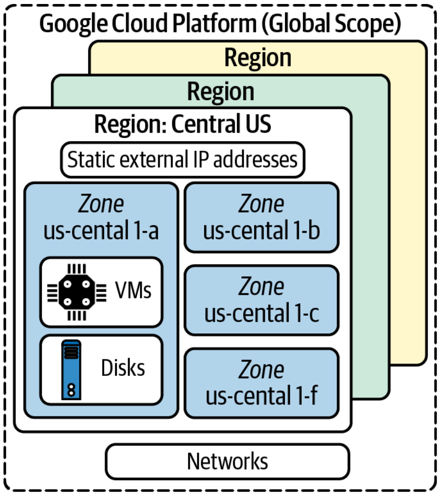


Figure 1-1. Compute and network resources in a region

In general, you should design production services to be deployed at least across a region, if not across multiple regions. This is often easier in Google Cloud than in other platforms, due to the large number of multiregional and global services, and the fact that these data centers are all connected with Google-owned fiber.

Projects

One of the key differences between Google Cloud and some other cloud platforms is the construct of a *project*. Every resource you create or interact with will exist in a project, and security roles and rules tend to be applied at a project level, though they can certainly be more granular. Resources within a project generally work together to form an application. A project naturally organizes resources securely, based on real-life use cases.

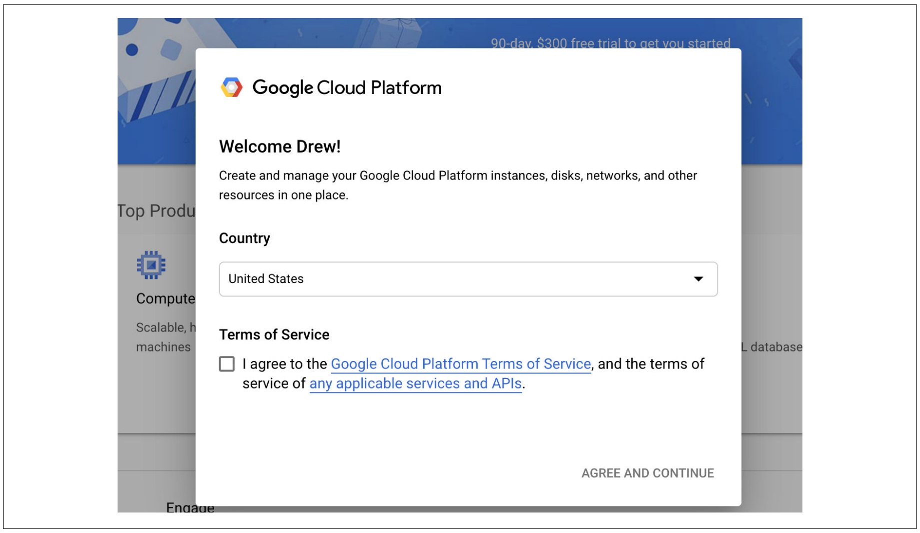
Often an application will have its own project for each life cycle, for example, “myapp-dev,” “myapp-staging,” and “myapp-prod.” Each successive project can have different security rules and a different copy of your application, with little work needed on your end to segment them. Sometimes projects are used at the team level. For example, a data science team primarily authoring data pipelines or performing data exploration will share a project for all this work, whereas the pipelines that get promoted to production will run in a separate, more locked-down project.

The majority of the time, most resources and services will interact with resources only in that project, and default security rules allow this. However, you can share resources between projects, and this becomes more common the larger your application or organization grows. For example, you may peer networks between projects or create security rules to allow objects stored on Google Cloud Storage (GCS) to be accessed across projects. Crossing a project boundary usually requires a conscious setup step, which helps with security.

How to create your first google cloud project

Here, we’ll set up an empty Google Cloud project and get $300 in credits.

1. Go to the Getting Started page and log in with a valid Gmail account.
2. Accept the terms of service, as shown in Figure 1-2.



Welcome Drew!

Create and manage your Google Cloud Platform instances, disks, networks, and other resources in one place.

Country

United States

Terms of Service

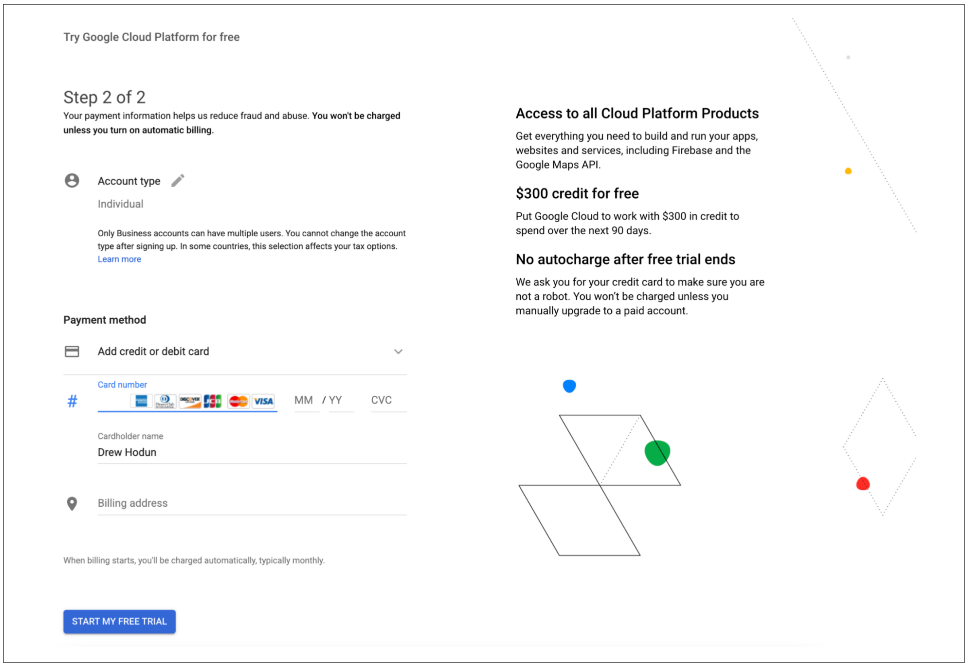
☐ I agree to the Google Cloud Platform Terms of Service, and the terms of service of any applicable services and APIs.

AGREE AND CONTINUE

Figure 1-2. Google Cloud Terms of Service dialog box

3. Activate your free trial with the button in the upper right to receive $300 in credits.

4. Add a credit card to get started, as shown in Figure 1-3. Don't worry; there is no autocharge or any extra charges after you go through your $300, so long as you turn off VMs and delete your data. It is generally hard to run up a huge bill, given your default quota for resources.



Try Google Cloud Platform for free

Step 2 of 2

Your payment information helps us reduce fraud and abuse. **You won't be charged unless you turn on automatic billing.**

Account type

Individual

Only Business accounts can have multiple users. You cannot change the account type after signing up. In some countries, this selection affects your tax options. Learn more

Payment method

Add credit or debit card

Card number MM / YY CVC

Cardholder name

Drew Hodun

Billing address

When billing starts, you'll be charged automatically, typically monthly.

START MY FREE TRIAL

Access to all Cloud Platform Products

Get everything you need to build and run your apps, websites and services, including Firebase and the Google Maps API.

$300 credit for free

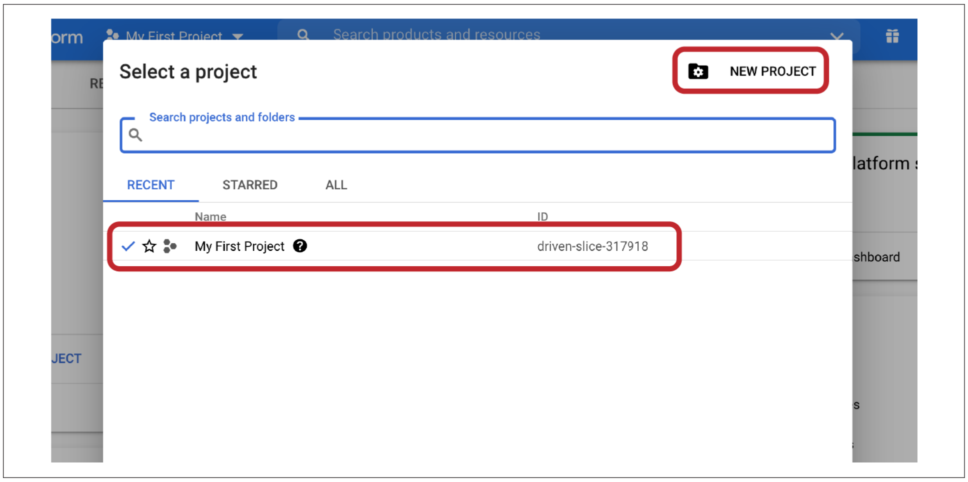
Put Google Cloud to work with $300 in credit to spend over the next 90 days.

No autocharge after free trial ends

We ask you for your credit card to make sure you are not a robot. You won't be charged unless you manually upgrade to a paid account.

Figure 1-3. Google Cloud credit card dialog box

You'll notice a new project has been created for you, as in Figure 1-4. You can decide to change the display name (but not the project ID or project number), or you can create a new project with the name you want.



My First Project driven-slice-317918

Figure 1-4. New project view

Congrats! You now have a project up and running on Google Cloud.

Enterprise projects

Google Cloud makes it easy to create one (or several) small projects for personal use, using a Gmail account. However, if you're working in an enterprise context, projects will belong to an organization (your company, for example) and be organized into folders (think departments). Almost all resources still belong to individual projects, with the exception of some security rules and network resources.

Your experience as a solo personal developer and as an enterprise developer will also feel very similar, although some more security rules and restrictions may be in place. For example, you may not be able to create projects at will the way you did with your Gmail account, and you won't have full power over those projects and resources. You will also likely see more errors related to organizational policies and restraints—for example, you shouldn't be exposing port 22 for SSH on all your VMs, and you'll see an error if you try. We recommend this guide to developing in a constrained Google Cloud environment.

Figure 1-5 is an example of a Google Cloud organization hierarchy, which organizes projects and resources.

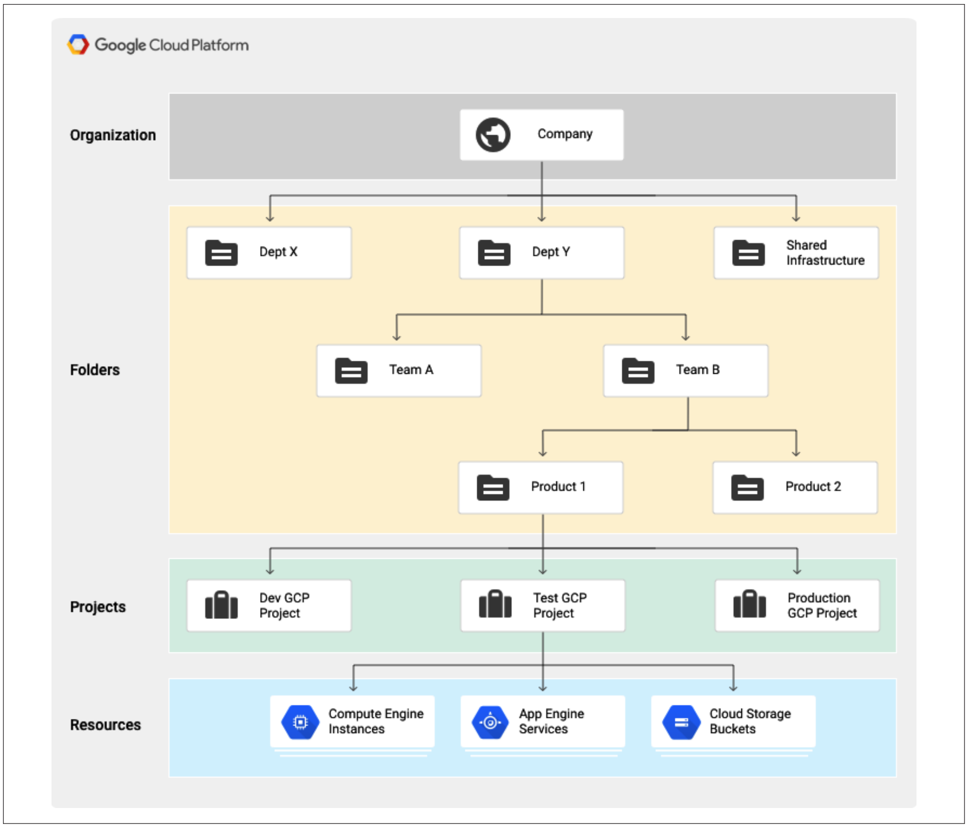
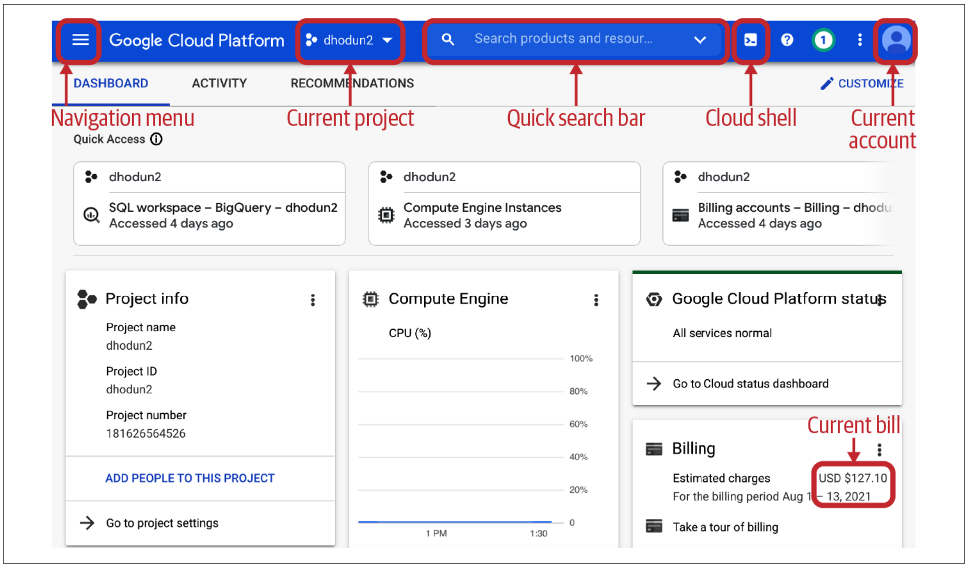


Figure 1-5. Google Cloud organization hierarchy (https://oreil.ly/eXevQ)

Cloud Console

Much of your day-to-day interaction with Google Cloud will be through the console via your browser. You've already seen the console, but you can again access it at the Google Cloud Platform landing page. Figure 1-6 shows the console with many of the more important items noted.



dhodun2
SQL workspace – BigQuery – dhodun2
Accessed 4 days ago

dhodun2
Compute Engine Instances
Accessed 3 days ago

dhodun2
Billing accounts – Billing – dhodu
Accessed 4 days ago

Project info
Project name
dhodun2
Project ID
dhodun2
Project number
181626564526
ADD PEOPLE TO THIS PROJECT
Go to project settings

Compute Engine
CPU (%)
100%
80%
60%
40%
20%
0
1 PM
1:30

Google Cloud Platform status
All services normal
Go to Cloud status dashboard

Billing
Estimated charges
USD $127.10
For the billing period Aug 1 – 13, 2021
Take a tour of billing

Figure 1-6. The Google Cloud Console

Table 1-2 shows some of the most commonly used parts of the Google Cloud Console.

Table 1-2. Parts of the Google Cloud Console

Name	Description
Navigation Menu	Click here to access the many GCP services.
Current Project	If you're working in a few different projects, it's helpful to see which one you're in.
Quick Search Bar	As much as I used to pride myself on knowing where all the services were listed in the service menu and pinning them for quicker access, I find myself using the search bar more and more to get to different pages quickly. Google is a search company after all.
Cloud Shell	Quickly open up a shell in your browser with gcloud and other tools installed. More on that in a bit.
Current Account	This is where you can easily see which account you're logged in as—particularly helpful if you are switching between your corporate account and Gmail account.
Current Project Bill	This is where your current month-to-date charges on the project appear.

Click the navigation menu to see the available services. You can click Compute Engine directly if you want to go straight to the default page (a list of your VMs in that project), or you can hover over Compute Engine and see the additional subpages you can directly access, as shown in Figure 1-7.

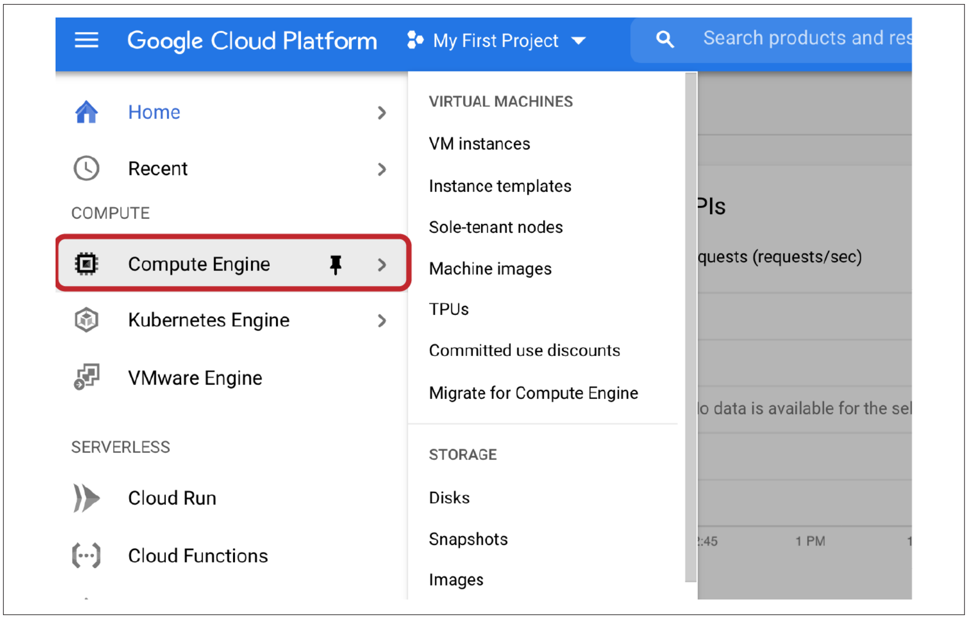


Figure 1-7. The navigation menu, where you access a service's main page and subpages

Note that you can pin individual services if you use them frequently so they show up at the top of the navigation menu, as shown in Figure 1-8.

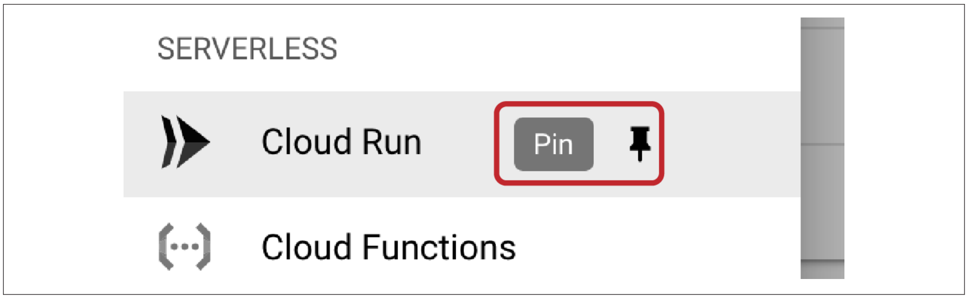


Figure 1-8. Pinning services in the navigation menu

Or you can use the search bar, which is often faster if you use many services.

Now, let's show you a few things you can do in the Cloud Console. First, we'll upload a file, and then we'll run a query.

This example shows how to use the console to perform simple, day-to-day actions that we have previously performed using command-line interface (CLI) tools. In this case, we'll create a GCS bucket and upload a file to it:

1. From the Cloud Console, open the Cloud Storage section, as in Figure 1-9.

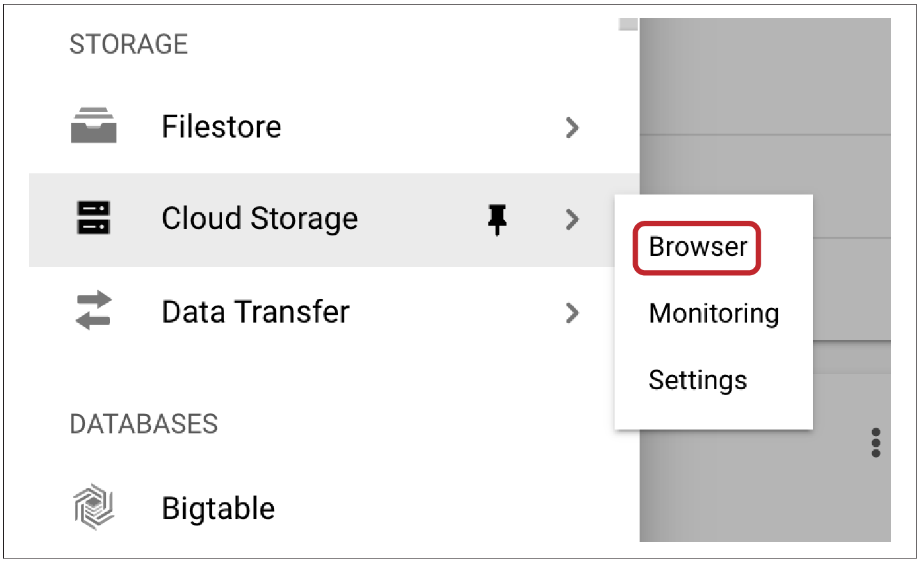
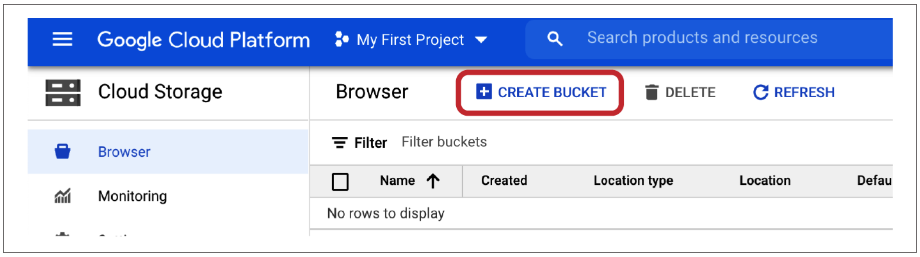


Figure 1-9. Cloud Storage browser from the navigation menu

2. Create a new bucket, as shown in Figure 1-10. Call it whatever you like.

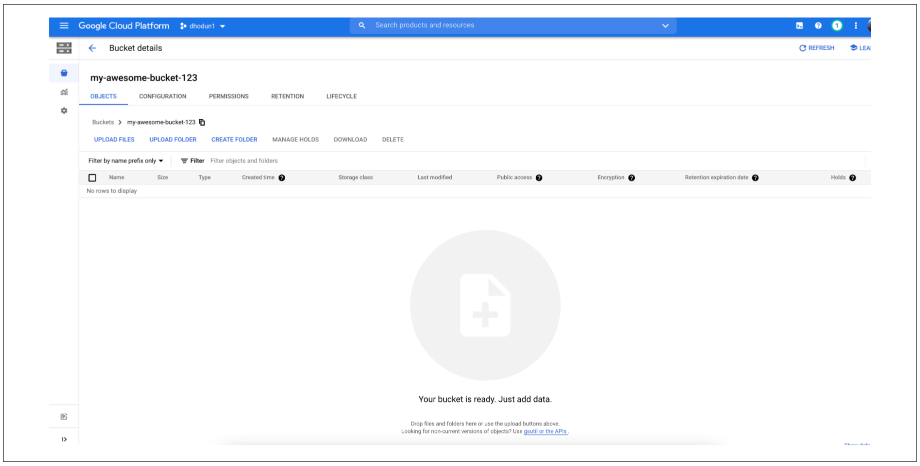


Name ↑	Created	Location type	Location	Defau

No rows to display

Figure 1-10. Create a new bucket

3. You should see your new bucket, as in Figure 1-11.



Name	Size	Type	Created time	Storage class	Last modified	Public access	Encryption	Retention expiration date	Holds

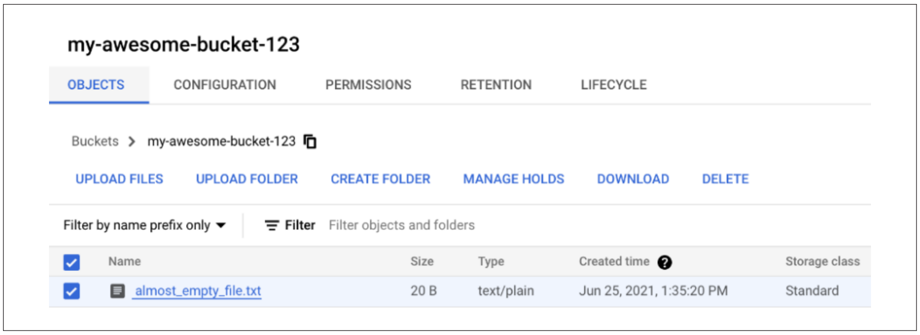
No rows to display

Your bucket is ready. Just add data.

Drop files and folders here or use the upload buttons above.
Looking for non-current versions of objects? Use gsutil or the APIs .

Figure 1-11. An empty bucket in the console

4. There are many ways to upload files to Cloud Storage. For now, create an empty text file or document on your desktop and then upload it through your browser. The result will look like Figure 1-12.



✓	Name	Size	Type	Created time	Storage class
✓	almost_empty_file.txt	20 B	text/plain	Jun 25, 2021, 1:35:20 PM	Standard

Figure 1-12. A new file in the new bucket

5. Congrats! You've uploaded a file. You can click it to see more data about the file, the size, and even an authenticated download link.
6. Don't forget to delete the object and the bucket to avoid long-term charges.

gCloud Command-Line Tool

Much of your work interacting with Google Cloud, whether creating VMs, deploying apps, or running BigQuery queries, can easily be done from your workstation, using the gcloud suite of utilities. While every Google Cloud service has an API and often a client library in your preferred language, gcloud is a hugely useful, common, and first-class way to interact with Google Cloud.

Installing gcloud

The client software development kit (SDK) (gcloud) is an easy way to get started quickly with these services from the CLI and through scripting. Your system will need Python installed. If you have any issues with installation, skip ahead to the "Cloud Shell" section for an easy, browser-based gcloud experience.

1. Head to the SDK installer documentation to download the gcloud client for your OS. Your install sequence will look something like this (for macOS):

   ```
   curl https://sdk.cloud.google.com | bash
   ```

2. Choose a directory (usually home directory). Just press Enter for default.
3. Allow the installer to modify your $PATH environment variable so you can immediately use gcloud just by typing it. Press **y** to continue.
4. Update your *.bashrc* or *.bash_profile* file so that the gcloud directory is always added to the `$PATH` environment variable. Leave it blank and press Enter to keep the default.
5. Restart your shell:

   ```
   exec -l $SHELL
   ```

6. Initiate your gcloud installation:

   ```
   gcloud init
   ```

7. You will need to choose an account (which may involve exiting to the browser to authenticate and copy and paste a code).
8. Pick a default project:

   ```
   You are logged in as: [dhodun@google.com].
   Pick cloud project to use:
    [1] dhodun1
    [2] dhodun2
   Please enter numeric choice or text value (must exactly match list
   item):
   ```

You will also have the choice of a default region or zone. These are common and helpful, particularly for VMs (in Google Compute Engine), especially if you are mainly working in a single region or zone and don't want to specify those as flags on every gcloud command.

gCloud commands

Now that you have gcloud installed, here are a series of useful commands you'll use. I recommend trying all of them now to get a feel for the tool.

Most gcloud commands follow the same format:

```
`gcloud` + `service` + `resource in that service` + `command`
```

Common commands are `list`, `create`, `delete`, and so on.

`gcloud config list`
: This lists your current gcloud config, including the currently set project, account, region, and zone:

```
dhodun@cloudshell:~ (dhodun2)$ gcloud config list
[compute]
region = us-central1
zone = us-central1-c
[core]
account = dhodun@google.com
disable_usage_reporting = True
project = dhodun2
```

`gcloud auth login`
: This adds another credentialed account to gcloud, which kicks off the browser-based auth workflow.

`gcloud auth list`
: This lists all the accounts currently logged in to gcloud. This is common if you have a corporate account and a personal account (and can somehow be logged in on both on the same system) or if you have a couple of personal accounts for various projects.

```
dhodun@cloudshell:~ (dhodun2)$ gcloud auth list
 Credentialed Accounts
ACTIVE ACCOUNT
*       dhodun@google.com
        dhodun@mycorp.com
```

`gcloud config set project <project_ID>`
: This command sets your currently configured default project for gcloud. Many other gcloud commands and even client libraries will infer the project to point at based on how gcloud is configured. Often, it is more convenient to set your default project with this command than to append "-p <project_ID>" to every other gcloud command you're running.

`gcloud auth set account <your_name@gmail.com>`
: This enables you to change accounts when a few of them are authenticated in gcloud.

`gcloud components update`
: This updates gcloud. Remember to do this somewhat frequently; new commands, services, and capabilities are always being launched on Google Cloud.

`gcloud compute instances create my-new-vm --image-family ubuntu-1604-lts --image-project ubuntu-os-cloud`
: This creates a new Ubuntu VM with all the other default settings in your project. Output:

```
Created [https://www.googleapis.com/compute/v1/projects/dhodun2/zones/us-
central1-c/instances/my-new-vm].
NAME       ZONE           MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTER-
NAL_IP   STATUS
my-new-vm  us-central1-c  n1-standard-1               10.128.0.29
35.223.11.69  RUNNING
```

`gcloud compute instances list`
: This lists all the VMs in your currently set project.

`gcloud container clusters list`
: This lists all the current Google Kubernetes Engine (GKE) clusters running in your project. You won't see anything if you haven't created a cluster yet.

`gcloud compute instances delete my-new-vm`
: This command deletes the same VM you created in the previous example. Press **y** to confirm.

`gcloud projects list --format="json"`
: You can add formatting and filtering patterns to any gcloud to make it more readable. This command shows what happens if you want to see your details for all your projects in pretty JSON format.

```
[
  {
    "createTime": "2021-08-06T11:02:22.009Z",
    "lifecycleState": "ACTIVE",
    "name": "dhodun2",
    "projectId": "dhodun2",
    "projectNumber": "181626564526"
  }
]
```

If you want a deeper dive into gcloud, check out the official cheat sheet.

Other gcloud Tools

Now that you have gcloud installed, you actually have a few other tools as well:

- gsutil for GCS (Google Cloud Storage) operations
- bq for CLI access to BigQuery, Google Cloud's serverless data warehouse
- kubectl, the open source tool to access and interact with Kubernetes clusters

These tools will be covered in future chapters, but here are a few sample commands to give you the flavor:

`gsutil ls`
: This lists all the GCS buckets you have access to.

`gsutil ls gs://gcp-public-data-landsat/`
: This is the normal list command on a folder in a bucket.

```
dhodun@cloudshell:~ (dhodun2)$ gsutil ls gs://gcp-public-data-landsat/
gs://gcp-public-data-landsat/index.csv.gz
gs://gcp-public-data-landsat/LC08/
gs://gcp-public-data-landsat/LE07/
gs://gcp-public-data-landsat/LM01/
gs://gcp-public-data-landsat/LM02/
gs://gcp-public-data-landsat/LM03/
gs://gcp-public-data-landsat/LM04/
gs://gcp-public-data-landsat/LM05/
gs://gcp-public-data-landsat/LO08/
gs://gcp-public-data-landsat/LT04/
gs://gcp-public-data-landsat/LT05/
gs://gcp-public-data-landsat/LT08/
```

`bq ls -project_id bigquery-public-data`
: This lists all the data sets you have list access to in the given project. Or leave the project flag blank to see everything you have access to in the current project. (If you have a new project, you won't see anything.)

```
dhodun@cloudshell:~ (dhodun2)$ bq ls -project_id bigquery-public-data
               datasetId
 -------------------------------------
  austin_311
  austin_bikeshare
  austin_crime
  austin_incidents
  Austin_waste
...
```

```
bq query --nouse_legacy_sql \
        'SELECT
           COUNT(*)
         FROM
           `bigquery-public-data`.samples.shakespeare'
```

This runs a query on a BigQuery data set, in this case simply counting the number of rows on a publicly available table. You'll see this output:

```
dhodun@cloudshell:~ (dhodun2)$ bq query --nouse_legacy_sql \
> 'SELECT
>    COUNT(*)
>  FROM
>    `bigquery-public-data`.samples.shakespeare'

Waiting on bqjob_r5dc0d7cca7a2a56_0000017b2be329b3_1 ... (0s) Current sta-
tus: DONE
+--------+
|  f0_   |
+--------+
| 164656 |
+--------+
```

`gcloud container clusters get-credentials CLUSTER_NAME --zone ZONE`

Authenticate and store credentials to a Google Kubernetes Engine cluster so you can then use the kubectl Kubernetes CLI. You can see in the output that the credentials are cached and a kubeconfig entry generated so that you can then use kubectl naturally.

```
dhodun@cloudshell:~ (dhodun2)$ gcloud container clusters get-credentials my-
first-cluster-1 --zone us-central1-c
Fetching cluster endpoint and auth data.
kubeconfig entry generated for my-first-cluster-1.
```

`kubectl get pods`

Now you can use kubectl to interact with your cluster like you would any Kubernetes clusters. You'll use this in Chapter 6.

```
dhodun@cloudshell:~ (dhodun2)$ kubectl get pods
NAME                                                READY   STATUS
RESTARTS   AGE
Redis                                               3/3     Running
0          2m1s
web-server                                          3/3     Running
0          79s
```

Cloud Shell

One of the great features for getting started with Google Cloud is Cloud Shell, a browser-based development and operations environment. It is essentially a just-in-time small VM that boots with gcloud, kubectl, client libraries, and other tools installed and is preauthenticated to Google Cloud with your account. It even has a web-based integrated development environment (IDE). This allows you to operate against your cloud environment without having to install any software on your desktop. It also can be used to log on to VMs in your browser, which prevents the need for downloading and managing SSH keys.

Here's how to connect to a VM, using Cloud Shell:

1. From the Cloud Console, click the Cloud Shell icon in the upper right, as in Figure 1-13.

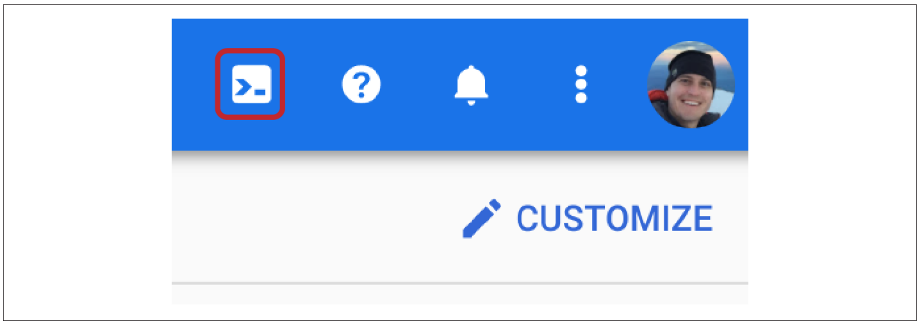
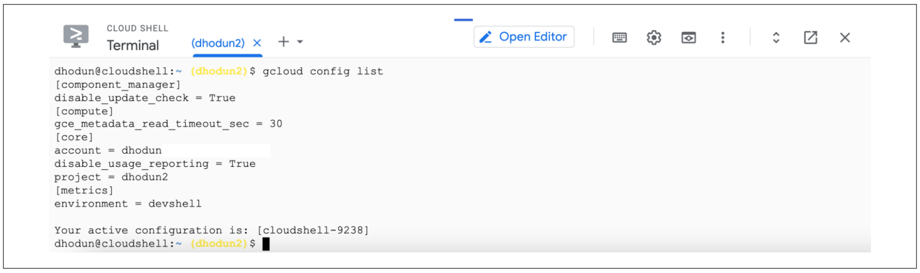


Figure 1-13. Cloud Shell icon

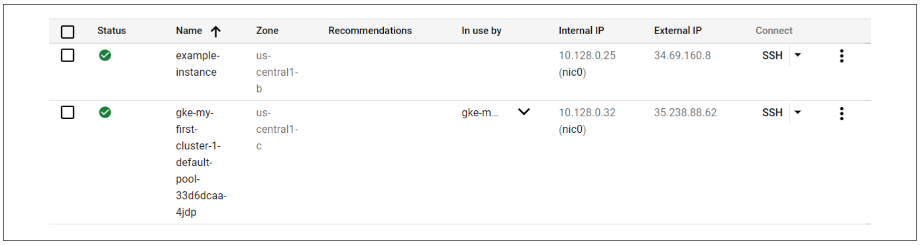
2. Once it is provisioned, run **`gcloud config list`** as in Figure 1-14 to see that you're authenticated, pointing to your current project, and ready to run other commands.



```
dhodun@cloudshell:~ (dhodun2)$ gcloud config list
[component_manager]
disable_update_check = True
[compute]
gce_metadata_read_timeout_sec = 30
[core]
account = dhodun
disable_usage_reporting = True
project = dhodun2
[metrics]
environment = devshell

Your active configuration is: [cloudshell-9238]
dhodun@cloudshell:~ (dhodun2)$
```

Figure 1-14. Basic Cloud Shell output

3. You can also use Cloud Shell from the GCE page to use SSH directly on a VM. If you haven't already deleted that test VM, you can go to the Google Compute Engine page and click the SSH button to the right of the VM to test this, as in Figure 1-15.

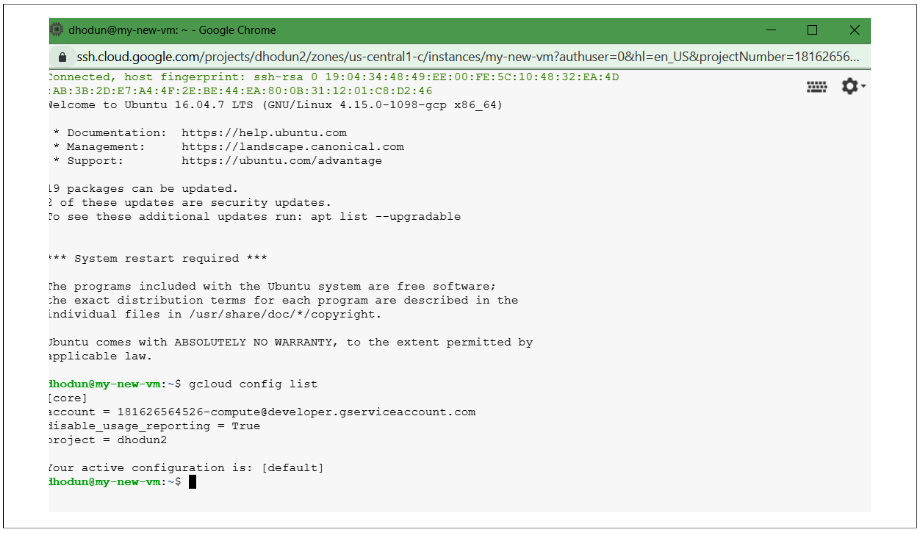


	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect	
☐	✓	example-instance	us-central1-b			10.128.0.25 (nic0)	34.69.160.8	SSH ▾	⋮
☐	✓	gke-my-first-cluster-1-default-pool-33d6dcaa-4jdp	us-central1-c		gke-m... ⌄	10.128.0.32 (nic0)	35.238.88.62	SSH ▾	⋮

Figure 1-15. The Cloud Shell button in the GCE VM list

4. A new window opens, and fresh SSH keys will be transferred to the VM so you can securely connect. Note: you don't have to worry about transferring these keys yourself!

5. Now that you're on the VM, as in Figure 1-16, you can verify that you have gcloud access. However, you will be authenticated as the default service account and not as your user account. Putting your user credentials on VMs is not recommended, because anyone who gains access to that machine could impersonate you when calling other Google Cloud APIs.



dhodun@my-new-vm: ~ - Google Chrome
ssh.cloud.google.com/projects/dhodun2/zones/us-central1-c/instances/my-new-vm?authuser=0&hl=en_US&projectNumber=18162656...

```
onnected, host fingerprint: ssh-rsa 0 19:04:34:48:49:EE:00:FE:5C:10:48:32:EA:4D
AB:3B:2D:E7:A4:4F:2E:BE:44:EA:80:0B:31:12:01:C8:D2:46
elcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-1098-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

9 packages can be updated.
 of these updates are security updates.
o see these additional updates run: apt list --upgradable


** System restart required ***

he programs included with the Ubuntu system are free software;
he exact distribution terms for each program are described in the
ndividual files in /usr/share/doc/*/copyright.

buntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
pplicable law.

hodun@my-new-vm:~$ gcloud config list
core]
ccount = 181626564526-compute@developer.gserviceaccount.com
isable_usage_reporting = True
roject = dhodun2

our active configuration is: [default]
hodun@my-new-vm:~$ █
```

Figure 1-16. Cloud shell SSH session into a GCE VM

Client Libraries

As you write increasingly cloud-native applications, you will interact with more and more cloud services in your code. For example, you may store files on Google Cloud Storage or write messages to the Pub/Sub message bus. Rather than write directly to the API, Google Cloud has client libraries in many languages to make this much easier, including Go, Java, Node.js, Python, Ruby, PHP, C#, and C++. Often they are shipped for individual services or groups of services. You can see all of them in the Cloud Client Libraries.

Here is a very simple example of using a client library versus using the gcloud and similar CLI commands we've used before. We will create a bucket with the Storage Client Library and Cloud Shell. We'll use a Python script and the client library to create a Google Cloud Storage bucket and upload a file.

This code sample, as well as all other code samples for this chapter, is in this book's GitHub repository. You can follow along and copy the code for this example.

1. Open a Cloud Shell environment as we just did.
2. Click Open Editor in the upper right of Cloud Shell to open the Cloud Shell IDE, as in Figure 1-17.

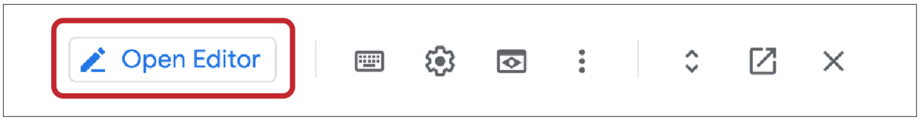


Figure 1-17. Open Editor button

3. In the File explorer, click File and then New File; and name it *storage_script.py*. Enter the following code to this file. Change the bucket name to something unique and random. Your project name is a good option. Save the file. You can also find the script in the GitHub repo, which you can clone into your Cloud Shell environment.

```
from google.cloud import storage

BUCKET_NAME = "INSERT-NEW-BUCKET-NAME"
FILE = "file.txt"

client = storage.Client()

# create a new bucket
bucket = client.bucket(BUCKET_NAME)
bucket.storage_class = "STANDARD"
new_bucket = client.create_bucket(bucket, location="us")

# get the bucket (useful if it already exists)
```

```
bucket = client.get_bucket(BUCKET_NAME)

# create and upload a new blob from text
blob = bucket.blob('remote/path/new_file.txt')
blob.upload_from_string('New file!')
print(blob.download_as_string())

# create and upload a new blob from file
blob2 = bucket.blob('remote/path/file.txt')
blob2.upload_from_filename(filename='file.txt')
print(blob2.download_as_string())
```

4. Create another file and name it **file.txt**. Add some text like **"Hello World!"** and save the file.
5. When you finish editing, click Open Terminal on the right side of Cloud Shell, as in Figure 1-18, to go back to terminal mode.

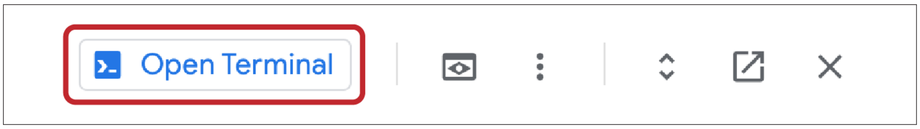


Figure 1-18. Open Terminal button in Cloud Shell

6. Before you run your script, you'll need to use the pip tool to install the Cloud Storage client library for Python from PyPi, Python's package repository.

```
pip3 install google-cloud-storage
```

7. Now you can run your script.

```
python3 storage_script.py
```

You should see the following output from the script reading the contents of both files:

```
b'New file!'
b'Hello World!'
```

8. Delete your files and the bucket, using the browser to avoid extra charges.

You just used one of the client libraries to connect programmatically to a Google Cloud service. There are many client libraries, depending on the services you're working with.

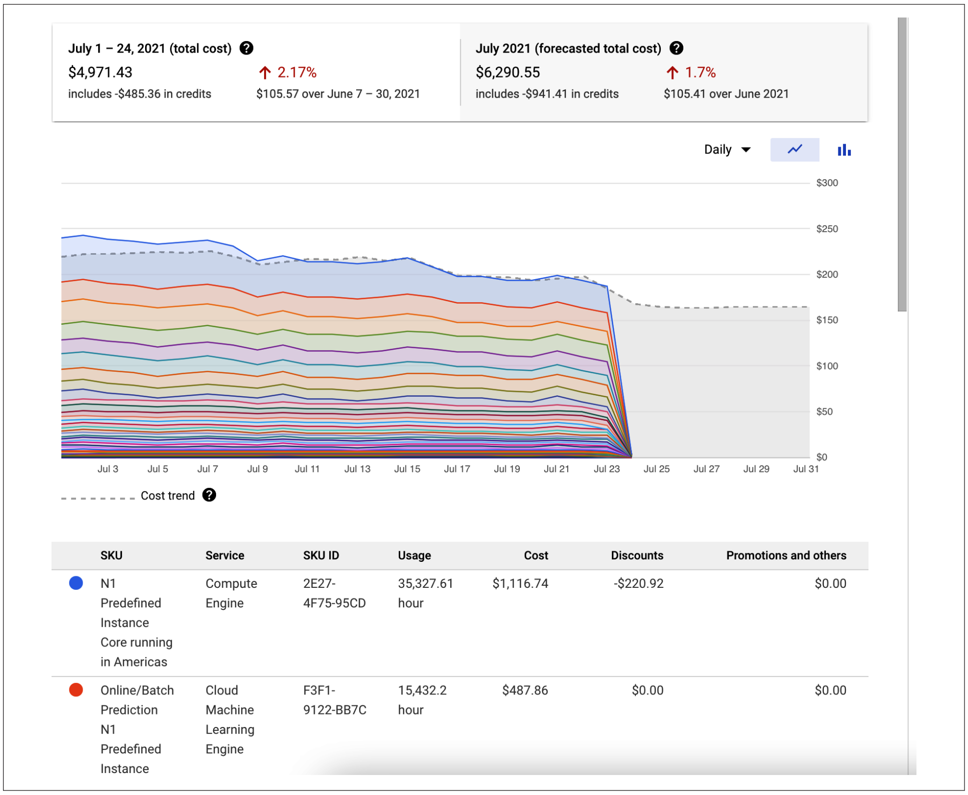
If you are working more on the infrastructure side, you will largely use the CLI tools or infrastructure workflow tools like Terraform, but if you are on the application side, you will frequently be using these client libraries.

Also note that when we created the client, `client = storage.Client()`, we didn't provide any credentials or a project ID. This is because the client libraries can infer credentials based on a number of settings on the local machine, starting with Application Default Credentials. In Cloud Shell, this has already been set for you to your user account.

This is beyond the scope of the intro, but just know that when you use your own credentials as Application Default Credentials, any code running on your machine that hits a Google Cloud API will try to use your potentially powerful user credentials. Refer to this guide for more.

Billing

While advanced setup of enterprise billing accounts is outside the scope of this book, you should understand the basics of billing. As you consume Google Cloud resources in a project, charges are accumulated and usually updated in the console nightly. If you open the Billing section of the console, you will see a summary of your current charges, as in Figure 1-19.



July 1 – 24, 2021 (total cost)
$4,971.43 — ↑ 2.17%
includes -$485.36 in credits — $105.57 over June 7 – 30, 2021

July 2021 (forecasted total cost)
$6,290.55 — ↑ 1.7%
includes -$941.41 in credits — $105.41 over June 2021

	SKU	Service	SKU ID	Usage	Cost	Discounts	Promotions and others
●	N1 Predefined Instance Core running in Americas	Compute Engine	2E27-4F75-95CD	35,327.61 hour	$1,116.74	-$220.92	$0.00
●	Online/Batch Prediction N1 Predefined Instance	Cloud Machine Learning Engine	F3F1-9122-BB7C	15,432.2 hour	$487.86	$0.00	$0.00

Figure 1-19. Billing breakdown for a single project

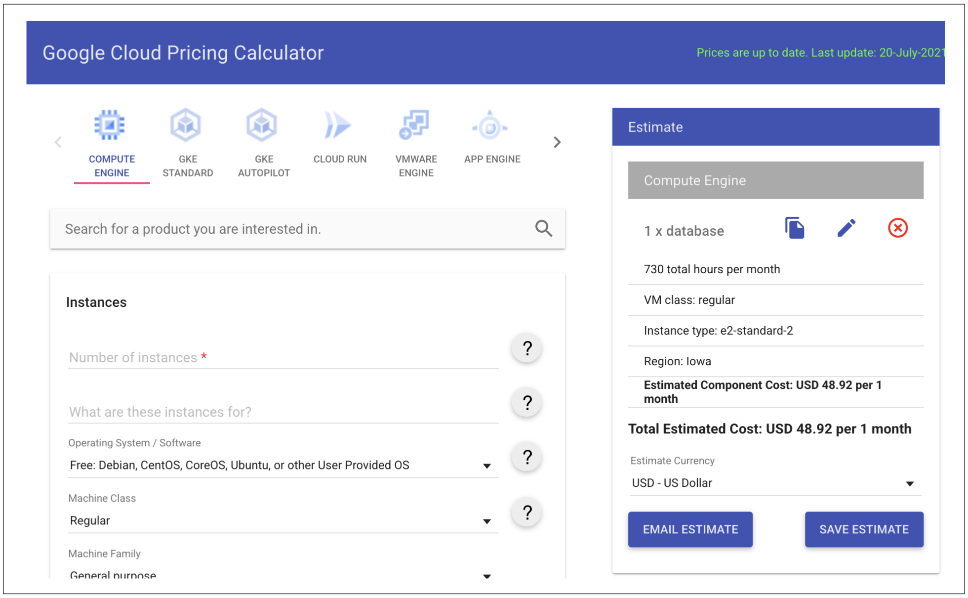
A project is always assigned to a billing account, which pays for the project. If you created a project on your credit card, you also created a billing account. In an enterprise setting, you will likely be given a project with a billing account already assigned. The finance or procurement team may run this account, allowing them control of the payment process, and you control of your project resources. You can still see the current cost of your project, however.

Pricing

Google Cloud services are charged as you use them. That is, there are no up-front costs, and services are often billed by the second or even less. If you have a VM powered on for 95 seconds, you pay for only 95 seconds of compute usage.

Each service has a different pricing model. For example, in BigQuery you pay for the underlying storage based on how much data you have and how long it is stored, and then you also pay for each query based on how much data is scanned. In Google Compute Engine, however, you pay per second for resources consumed for powered-on VMs, such as cores, RAM, and disk storage.

If you just created an account, you have $300 in free credits to use for a few months. Many services also have a free tier for small usage to help you get started. Check out documentation for each service for more details. A great tool to understand and predict your charges is the Google Cloud Pricing Calculator. You can add services to estimate your bill, as in Figure 1-20.


Google Cloud Pricing Calculator
Prices are up to date. Last update: 20-July-2021
COMPUTE ENGINE
GKE STANDARD
GKE AUTOPILOT
CLOUD RUN
VMWARE ENGINE
APP ENGINE
Search for a product you are interested in.
Instances
Number of instances *
What are these instances for?
Estimate
Compute Engine
EMAIL ESTIMATE
SAVE ESTIMATE

Operating System / Software: Free: Debian, CentOS, CoreOS, Ubuntu, or other User Provided OS

Machine Class: Regular

Machine Family: General purpose

1 x database

730 total hours per month

VM class: regular

Instance type: e2-standard-2

Region: Iowa

Estimated Component Cost: USD 48.92 per 1 month

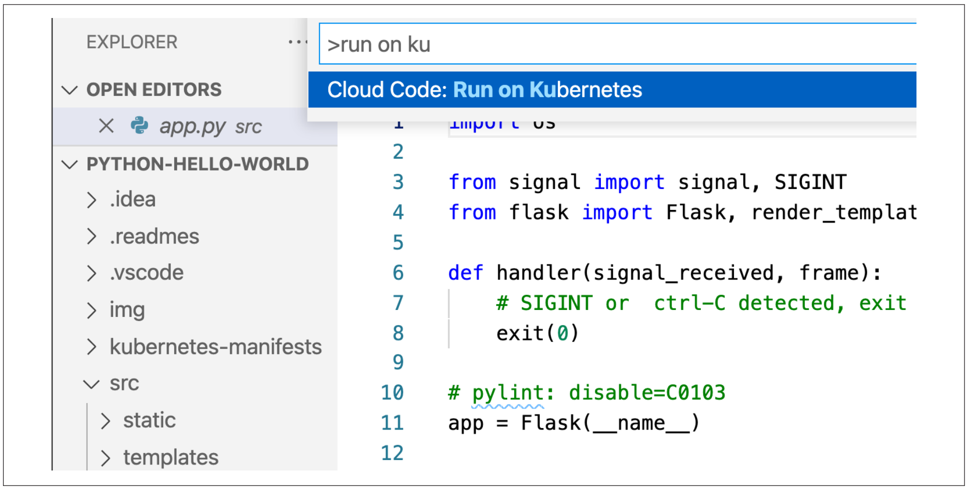
Total Estimated Cost: USD 48.92 per 1 month

Estimate Currency: USD - US Dollar

Figure 1-20. Google Cloud Pricing Calculator

Cloud Code (IDE Extensions)

Another tool absolutely worth installing is Cloud Code, a plug-in for popular IDEs, VSCode, and IntelliJ as well as Cloud Shell. Cloud Code provides auto-completion and in-IDE tooling for Kubernetes and Cloud Run–based applications. It bridges the gap between local development and deployment in cloud environments, provides live debugging for your container workloads, and allows you to manage these environments with a single click. Figure 1-21 shows easily running an application from the IDE, targeting a local Kubernetes minikube cluster. You can just as easily target a cluster in the cloud.



```
import os

from signal import signal, SIGINT
from flask import Flask, render_templat

def handler(signal_received, frame):
    # SIGINT or  ctrl-C detected, exit
    exit(0)

# pylint: disable=C0103
app = Flask(__name__)

```

Figure 1-21. Cloud Code run options

Moving from Another Cloud

If you are moving to Google from another cloud, you will find a number of very helpful guides to translate your existing knowledge of tools, products, services, and architectures to the Google Cloud world. We highly recommend exploring these early in your learning:

- Google Cloud for AWS Professionals
- Google Cloud for Azure Professionals
- Azure and AWS to Google Cloud Service Map

This Cookbook

The rest of this book is divided based on technology or product areas. Specific chapters, for example, cover topics like Kubernetes and BigQuery. The rest of these chapters are also divided into the typical cookbook format, with a problem to be solved and a solution, using cloud tools, accompanied by example code or scripts. Now, on to Chapter 2, which covers Google Cloud Functions.

CHAPTER 2
Cloud Functions

Google Cloud Functions is a serverless compute platform that allows you to run your code based on triggers such as uploading files to Cloud Storage or adding/deleting/updating Firestore documents, or trigger your functions directly from the web. Google Cloud provides two types of Cloud Functions: HTTP functions and background functions. You invoke HTTP functions from standard HTTP requests. You use background functions to trigger your function based on an event such as files being uploaded to Cloud Storage, receiving a message in Pub/Sub, updating a Firestore collection, or basing it on a schedule.

Google Cloud Functions abstract the compute infrastructure and allow you to focus on your code. You don't have to worry about patching operating systems or provisioning resources. Cloud functions scale automatically; they can scale from a single invocation to millions without intervention from the developer. In this chapter, we will present a range of recipes, from introductory recipes you can use to send emails or respond to SMS messages to advanced recipes that show you how to integrate CI/CD into your development workflow and integrate with Cloud Endpoints for API management.

Google Cloud Functions can be written in Node.js, Python, Go, Java, .NET, Ruby, and PHP programming languages. In this chapter, you will be using Node.js. All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.

3. Installed and configured gcloud, as described in Chapter 1.
4. Enabled the Cloud Functions and Cloud Build API:

   ```
   gcloud services enable cloudfunctions.googleapis.com
   gcloud services enable cloudbuild.googleapis.com
   ```

You will be creating many cloud functions in this chapter. If you do not want to run up the costs, don't forget to delete the cloud function if you're not using it any longer:

```
gcloud functions delete NAME --region=REGION
```

2.1 Creating a Public HTTP Google Cloud Function

Problem

You have a simple Hello World Node.js application that you need to host in the cloud.

Solution

Leverage Node.js to create an HTTP cloud function to use the response object to send a short Hello World response.

1. On your local workstation, create a temporary folder to hold the files you will need to create the Hello World HTTP function.
2. In your terminal, run the following command in the temporary folder you created:

   ```
   npm init
   ```

3. Accept the defaults when prompted to set up a new npm package.
4. In your favorite IDE, create an *index.js* file in the root of the directory you created in step 1 and copy the following code to the file:

   ```
   exports.helloHttp = (req, res) => {
     res.send(`Hello World!`);
   };
   ```

5. To deploy the cloud function, run the following command:

   ```
   gcloud functions deploy hello-http-function --entry-point helloHttp --
   runtime nodejs12 --trigger-http
   ```

6. When presented with the choice to allow the function to be unauthenticated, which allows anonymous access (`[helloHttp]? (y/N)`), select Y. Selecting Y allows users to access your function without authentication, or you can `-allow-unauthenticated` when running the `gcloud functions deploy` command.

7. In the Google Cloud Console, from the navigation menu, choose Compute and then Cloud Functions.
8. Locate your newly deployed cloud function and select it.
9. Select the Trigger tab.
10. Click the Trigger URL to test your newly deployed function in a new browser tab.

Discussion

You have successfully deployed your first HTTP Google cloud function that allows all users to access it. It is a simple "Hello World" but provides you some basic concepts on deploying functions. Let's break down the deployment process further and understand the arguments passed to the `gcloud functions deploy` command:

```
gcloud functions deploy NAME --entry-point NAME --runtime RUNTIME TRIGGER
```

`NAME`
: The registered name of the cloud function you are deploying. NAME can only contain letters, numbers, underscores, and hyphens.

`--entry-point ENTRY-POINT`
: If you want to make your registered name different from the name of the function.

`--runtime RUNTIME`
: The name of the runtime you are using. Examples include:

  - *nodejs10*
  - *python37*
  - *go111*
  - *java11*

2.2 Authenticating an HTTP Google Cloud Function

Problem

You have a simple Hello World Node.js application; you want to authorize access to it.

Solution

Configure the cloud function to deny unauthenticated requests.

If you're following along after creating the first function in Recipe 2.1, you need to delete the first function and then redeploy without allowing unauthenticated access by selecting N when prompted.

The steps for creating a secure HTTP cloud function are the same for creating a public HTTP Google Cloud Functions recipe, with one difference: when you are prompted to allow unauthenticated `[helloHttp]? (y/N)`, you would select N. Selecting N restricts users to only those you have allowed to access your function.

You can disable the Yes/No prompt by including a -q flag when running the gcloud functions as follows:

```
gcloud functions deploy hello-http-function --entry-
point helloHttp --runtime nodejs10 --trigger-http -q
```

If prompts are disabled, the default values are used.

Discussion

At this point you have successfully created a Hello World cloud function that is restricted to the users you add and provided the cloud functions invoker role. To test, navigate to the Cloud Function HTTP URL. To find your URL in your Cloud Console, navigate to Cloud Functions and click the function you deployed. Click the TRIGGER tab. If you click the link, you will be denied access to the Cloud Function.

Since you did not provide credentials, the function denied you access. To securely access the function, you can test with your email address that you use for Google Cloud. Run the following command:

```
curl -X GET "[YOUR_CLOUD_FUNCTION_URL]" -H \
    "Authorization: Bearer \
    $(gcloud auth print-identity-token)" --header \
    "Content-Type: application/json"
```

The `gcloud auth print-identity-token` command is not recommended for production applications. Use service accounts for production applications. This account will print your identity token, which will be used to authorize access to the cloud function.

"Hello World" should print on the command line. By default, your user account to access Google Cloud has the Cloud Functions Invoker role. The cloudfunctions.invoker role can invoke an HTTP function, using its public URL, but can't perform any administrative actions on it. To allow additional users to invoke the function, perform the following steps:

1. In the Google Cloud Console, click the checkbox next to the function created in this recipe.
2. Click Permissions at the top of the screen. Click Add Member.
3. In the New Members field, enter the email address of the service account you want to grant access to.

4. From the ‘Select a Role’ drop-down menu, select Cloud Functions > Cloud Functions Invoker.
5. Click Save.

2.3 Accessing Environment Variables at Runtime

Problem

You need a way for your code at runtime to access key or value pairs as third-party API keys without having to hard-code the values in your code.

Solution

Create a cloud function with environment variables to specify arbitrary key or value pairs at the time of deployment. These key or value pairs will be surfaced and accessible by your code at runtime.

1. On your local workstation, create a temporary folder to hold the files you will need to create the HTTP function.
2. In your terminal, run the following command in the temporary folder you created:
   ```
   npm init
   ```
3. Accept the defaults when prompted to set up a new npm package.
4. In your favorite IDE, create an *index.js* file in the root of the directory you created in step 1 and copy the following code to the file:
   ```
   exports.helloHttp = (req, res) => {
     res.send(process.env.MY_MESSAGE);
   };
   ```
5. To deploy the cloud function, run the following command:
   ```
   gcloud functions deploy hello-http-function --entry-point helloHttp --
   runtime nodejs12 --trigger-http --set-env-vars MY_MESSAGE=""”Hello
   Earth"”
   ```
6. In the Google Cloud Console, from the Navigation Menu, choose COMPUTE and Cloud Functions.
7. Locate your newly deployed cloud function and select it.
8. Select the TRIGGER tab.
9. Click the Trigger URL to test your newly deployed function in a new browser tab.
10. You will see the message “Hello World,” which from your code retrieves the environment variable MY_MESSAGE and returns the string value of the key.

Discussion

You have successfully created an environment variable that holds a value to your assigned key. Environment variables are key/value pairs associated with the respective function you deployed and are not visible to other functions in your project.

2.4 Sending Emails from Cloud Functions with SendGrid

Problem

You need the ability to send emails programmatically from your applications by calling a secure REST API.

Solution

Leverage the SendGrid SDK for Node.js to send emails from Google Cloud Functions.

1. On your local workstation, create a temporary folder to hold the files you will need to create the `SendGrid HTTP` function.
2. In your terminal, run the following command in the temporary folder you created:
    ```
    npm init
    ```
3. Accept the defaults when prompted to set up a new npm package.
4. Run the following command to install the SendGrid package:
    ```
    npm install @sendgrid/mail -save
    ```
5. Get your code ready for the `SendGrid` function by creating the *index.js* file, and copy the following code:
    ```
    const sendgrid = require('@sendgrid/mail');
    exports.sendGrid = async (req, res) => {
        console.log('running sendGrid Function')
        try {
            if (req.method !== 'POST') {
                const error = new Error('Only POST requests are accepted');
                error.code = 405;
                throw error;
            }
            const msg = {
                to: req.body.to,
                from: req.body.from,
                subject: req.body.subject,
                text: req.body.text
            };
            sendgrid.setApiKey(process.env.SENDGRID_API_KEY);
    ```

```
        sendgrid.send(msg)
            .then((response) => {
                console.log(response)
                if (response.statusCode < 200 || response.statusCode >=
400) {
                    const error = Error(response.body);
                    error.code = response.statusCode;
                    throw error;
                }
                res.status(200).send(`\n\n Email Sent to ${req.body.to}
\n\n`);
            })
        return Promise.resolve();
    } catch (err) {
        console.error(err);
        const code =
            err.code || (err.response ? err.response.statusCode : 500)
|| 500;
        res.status(code).send(err);
        return Promise.reject(err);
    }
};
```

The following line in the code block that follows is where you will extract the SendGrid API key environment variable; you can reference Recipe 2.3 for more information on environment variables:

```
sendgrid.setApiKey(process.env.SENDGRID_API_KEY);
```

6. Deploy your cloud function and set unauthenticated to No.

```
gcloud functions deploy \
          sendGrid --entry-point sendGrid \
          --runtime nodejs12 --trigger-http \
          --allow-unauthenticated
```

7. To configure SendGrid, you will need to enable the SendGrid API in the Google Cloud Marketplace as well as retrieve your SendGrid API key.
   a. Go to Google Cloud Marketplace Solutions to sign up for the SendGrid email service.
   b. Once the SendGrid service has been enabled, you can retrieve the API key on the SendGrid website. The API key will authorize your application to send SMS messages through the SendGrid service. Click Manage API Keys on the SendGrid website.
   c. Within the SendGrid website, navigate to Settings > API Keys.

    d. Click Create API Key.
    e. Copy the API key created.

8. Set a cloud function environment variable (see Recipe 2.3 that will hold your SendGrid API key, name the key **SENDGRID_API_KEY**, and set the value to your SendGrid API key).
9. Set Google Cloud Application Default Credentials (ADC) locally. The following command obtains the user access credentials and puts them on your local workstation. This command is useful when you are developing code that would normally use a service account but need to run the code in a local development environment. Using ADC is not suggested for production environments. Review the best practices to authenticate applications securely in Google Cloud. Use the command:

   ```
   gcloud auth application-default login
   ```

10. To test your newly created function, run the following `curl` command and replace the values in brackets with your settings. Example: replace `[TO_EMAIL]` with `someone@example.org`. To find your cloud function URL, go to Google Cloud Console > Compute > Cloud Functions, select the newly deployed function, and select the TRIGGER tab:

    ```
    curl -X POST "[YOUR_CLOUD_FUNCTION_URL]" -H "Authorization: Bearer $
    (gcloud auth print-identity-token)"
    --data '{"to":"[TO_EMAIL]", "from":"[FROM_EMAIL]", "subject":"[YOUR_SUB-
    JECT]","text":"[YOUR_TEXT]"}'
    --header "Content-Type: application/json"
    ```

11. You will see the following message on terminal on a successful execution:

    ```
    Email Sent to [TO_EMAIL]
    ```

Discussion

At this point, you have successfully created a cloud function to send email with Send-Grid. You've also set the parameters in the `curl` request to whom you want to send the email, plus the email contents. To allow users, you will need to give the respective user who needs to use your cloud function the Cloud Functions Invoker role. You can follow the Authenticating End-Users recipe to add users.

2.5 Deploying Cloud Functions with a GitLab CI/CD Pipeline

Problem

You want an automated way of deploying cloud functions when committing your code to a Git repository.

Solution

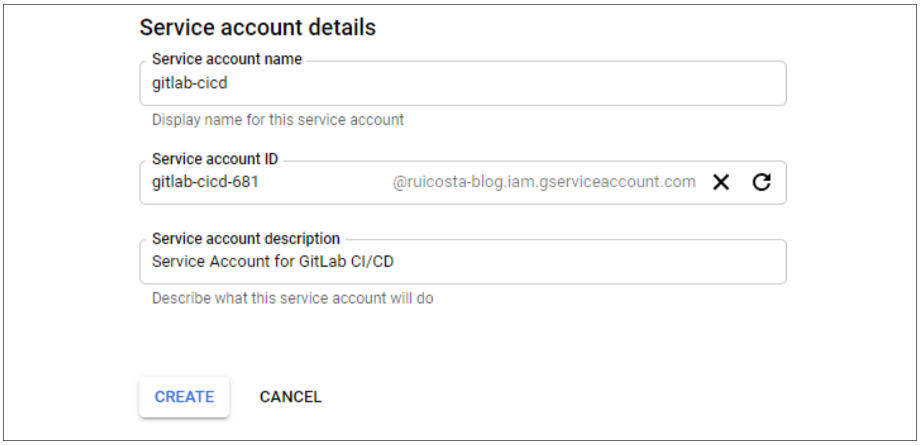
Leverage GitLab's CI/CD pipeline to automate your deployment. You will need the following prior to following the instructions:

- A GitLab account to get started.
- The GitLab documentation on creating a repository.
- An empty GitLab repository created and cloned locally to your workstation.

To authorize GitLab access to our Google Cloud project, we will need to create a Google Cloud service account and assign the required roles to the service account. This service account will authorize GitLab to deploy the cloud function to the defined project.

1. Open the IAM & Admin in the Google Cloud Console.
2. On the menu, click Service Accounts.
3. Click CREATE SERVICE ACCOUNT.
4. Enter your service account details:
   - Service account name
   - Service account ID
   - Service account description

   When completed, it should look like Figure 2-1.


Service account details

Service account name: gitlab-cicd

Display name for this service account

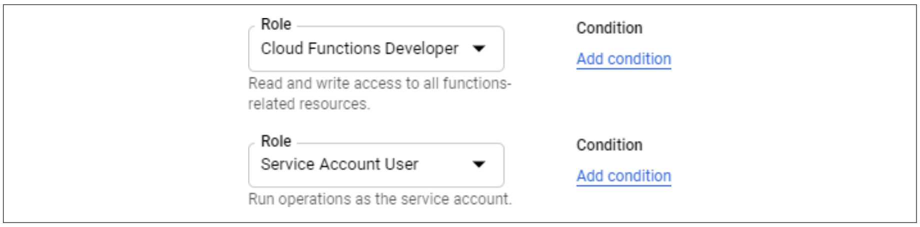
Service account ID: gitlab-cicd-681 @ruicosta-blog.iam.gserviceaccount.com

Service account description: Service Account for GitLab CI/CD

Describe what this service account will do

Figure 2-1. Service Account Details page

5. Click Create and Continue.
6. Assign the following roles to the service account. Figure 2-2 shows the roles selected:
   - Cloud Functions Developer
   - Service Account User


Role: Cloud Functions Developer

Read and write access to all functions-related resources.

Role: Service Account User

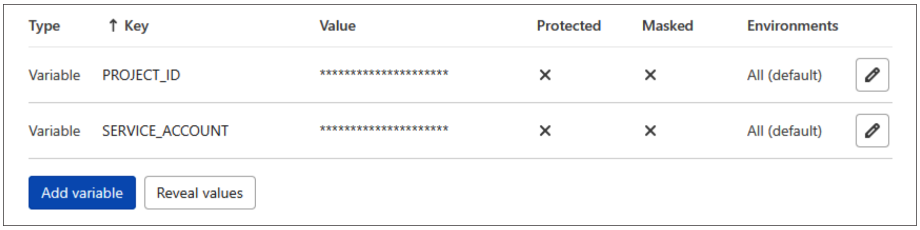
Run operations as the service account.

Figure 2-2. Selected roles

7. Click Continue.
8. Click Done.
9. Locate the newly created service account, click the Actions icon, and select Manage Keys.
10. Click ADD KEY and select Create New Key.
11. Choose JSON and click CREATE. This will download the JSON file to your local workstation. Review the best practices to authenticate applications securely in Google Cloud.
12. Head over to your GitLab project. In Settings, open CI/CD.
13. Click Expand Variables.

14. Create two new variables, labeled as:
    - PROJECT_ID
    - SERVICE_ACCOUNT

15. Enter your Google Cloud project ID.
16. Open the JSON file for the service account you downloaded before, and copy and paste its contents to the SERVICE_ACCOUNT key. It should look something like Figure 2-3.



Type	↑ Key	Value	Protected	Masked	Environments
Variable	PROJECT_ID	********************	✕	✕	All (default)
Variable	SERVICE_ACCOUNT	********************	✕	✕	All (default)

Figure 2-3. GitLab variables

At this point, you have the authorization configured for GitLab to deploy cloud functions to your Google Cloud project. The next step is to prepare your application code to be pushed to the main branch in your GitLab repository.

17. In your empty local GitLab repository run, accept all the defaults:

```
npm init
```

18. Get your code ready by creating an *index.js* file in the root of your cloned repository and copy the following code:

```
exports.helloHttp = (req, res) => {
  res.send(`Hello World!`);
};
```

19. Create a new *.gitlab-ci.yml* file in the root of your cloned repository.
20. Copy the following code to *.gitlab-ci.yaml*:

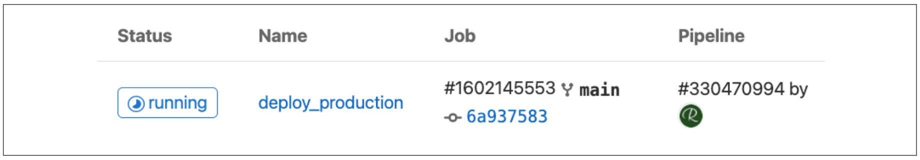
```
image: google/cloud-sdk:latest
stages:
  - deploy_production
deploy_production:
  stage: deploy_production
  only:
    - main
  script:
    - echo $SERVICE_ACCOUNT > ${HOME}/gcloud-service-key.json
    - gcloud auth activate-service-account --key-file ${HOME}/gcloud-
service-key.json
```

```
        - gcloud --quiet --project $PROJECT_ID functions deploy helloHttp --
    runtime=nodejs10 --trigger-http
```

21. Commit your changes and push the code to your repository, for example:

    ```
    git commit -a -m "My First Commit"
    git push origin main
    ```

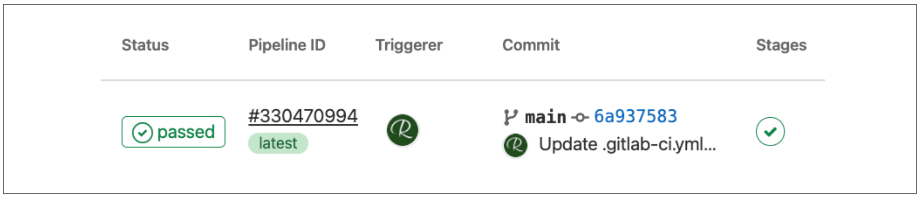
22. If you head back to GitLab CI/CD, you should see your job running as shown in Figure 2-4.



Status	Name	Job	Pipeline
running	deploy_production	#1602145553 main 6a937583	#330470994 by

Figure 2-4. Running GitLab job

23. You should see a similar output as Figure 2-5 once the pipeline has successfully executed.



Status	Pipeline ID	Triggerer	Commit	Stages
passed	#330470994 latest		main 6a937583 Update .gitlab-ci.yml...	✓

Figure 2-5. Completed GitLab job

24. Open the cloud function in the Google Cloud Console.
25. You should see the cloud function successfully deployed.

Discussion

Congratulations! You have successfully configured GitLab to deploy continuously to Google Cloud Functions when commits are performed on the main branch. Continuous delivery (CD) ensures the delivery of continuous integration (CI) validated code to your application by a deployment pipeline. In Recipe 2.7, you will learn how to perform CI to validate your code prior to deployment of your code to a production environment.

2.6 Responding to SMS Messages with Twilio and Cloud Functions

Problem

Your company is looking for a way to respond to client inquiries via SMS.

Solution

Create a cloud function to reply to an SMS message, using Twilio, which software to send and receive text messages.

1. On your local workstation, create a temporary folder to hold the files you will need to create the SMS HTTP function.
2. In your terminal, run the following command in the temporary folder you created:

   ```
   npm init
   ```

3. Accept the defaults when prompted to set up a new npm package.
4. Run the following command to install the SendGrid package:

   ```
   npm install twilio -save
   ```

5. Get your code ready for the SMS function by creating the *index.js* file and copy the following code:

   ```
   const twilio = require('twilio');
   const MessagingResponse = twilio.twiml.MessagingResponse;
   const projectId = process.env.GCLOUD_PROJECT;
   const region = 'us-central1';
   const TWILIO_AUTH_TOKEN = process.env.TWILIO_AUTH_TOKEN
   exports.reply = (req, res) => {
     let isValid = true;

     if (process.env.NODE_ENV === 'production') {
       isValid = twilio.validateExpressRequest(req, TWILIO_AUTH_TOKEN, {
         url: `https://${region}-${projectId}.cloudfunctions.net/reply`
       });
     }
     if (!isValid) {
       res
         .type('text/plain')
         .status(403)
         .send('Twilio Request Validation Failed.')
         .end();
       return;
     }
     const response = new MessagingResponse();
     response.message('Hello from the Google Cloud Cookbook.');
   ```

```
        res
          .status(200)
          .type('text/xml')
          .end(response.toString());
      ;
```

6. Deploy your HTTP cloud function:

   ```
   gcloud functions deploy \
       sendGrid --entry-point sendGrid \
       --runtime nodejs12 --trigger-http \
       --allow-unauthenticated
   ```

7. Create a Twilio account at *https://www.twilio.com/try-twilio*.
8. In your Twilio console, create a phone number.
9. Once you have a phone number assigned, click Manage Numbers.
10. Under Messaging:
    a. Set Configure With to Webhooks/TwiML.
    b. Set A Message Comes In to Webhook and enter the following URL:

       `https://us-central1-[YOUR_PROJECT_ID].cloudfunctions.net/reply`

    c. Click Save.
11. Return to the Twilio Account Settings.
12. Copy the Auth token for your live credentials. You will need it later in this recipe
13. Set a cloud function environment variable that will hold your Twilio Auth token, name the key **`TWILIO_AUTH_TOKEN`**, and set the key to your Twilio Auth token. Reference Recipe 2.3 on how to create environment variables.
14. Send an SMS message to your Twilio number.

You should receive a response with a message that has been defined in the cloud function.

Discussion

In this recipe, you configured a Twilio to trigger your cloud function via an HTTP request. Your cloud function, when triggered, sends a reply message defined in the code as `response.message('Hello from the Google Cloud Cookbook.')`. Twilio receives the response message and uses this response as the SMS payload. Twilio also provides you with robust documentation to get you started using its service quickly.

2.7 Unit Testing with GitLab and Cloud Functions

Problem

You need a method to perform testing on your code before it's deployed to production.

Solution

Use Mocha and the GitLab CI/CD to perform unit testing on your code prior to deployment to a production environment. Prior to following the instructions, you will need:

1. A GitLab account.
2. An empty GitLab repository created and cloned locally to your workstation.

To authorize GitLab access to our Google Cloud project, you will need to create a Google Cloud service account and assign the required roles to it. You can reference Recipe 2.5 on the steps to do this.

At this point, you have the authorization configured for GitLab to deploy cloud functions to your Google Cloud project. The next step is to prepare your application code to be pushed to the main branch in your GitLab repository.

1. In your empty GitLab repository local folder:
    a. Create a folder called *test* in the root folder of your GitLab repository local folder.
    b. Create index.test.js in the newly created test folder.
    c. Create a file *index.js* and .gitlab-ci.yml in the root folder of your GitLab repository local folder.
2. Copy the following code to the respective labeled files in the root of your locally cloned repository:

```
test/index.test.js
const assert = require('assert');
const sinon = require('sinon');
const uuid = require('uuid');
const {helloHttp} = require('..');
it('helloHttp: should print a name', () => {
  // Mock ExpressJS 'req' and 'res' parameters
  const name = uuid.v4();
  const req = {
    query: {},
    body: {
      name: name,
    },
  };
  const res = {send: sinon.stub()};
  // Call tested function
  helloHttp(req, res);
  // Verify behavior of tested function
  assert.ok(res.send.calledOnce);
  assert.deepStrictEqual(res.send.firstCall.args, [`Hello ${name}!`]);
});

.gitlab-ci.yml

image: google/cloud-sdk:latest
stages:
  - test
  - deploy_production
test:
  stage: test
  script:
    - npm install
    - npm run test
deploy_production:
  stage: deploy
  only:
    - main
  script:
    - echo $SERVICE_ACCOUNT > ${HOME}/gcloud-service-key.json
    - gcloud auth activate-service-account --key-file ${HOME}/gcloud-
service-key.json
    - gcloud --quiet --project $PROJECT_ID functions deploy helloHttp --
runtime=nodejs10 --trigger-http

index.js
exports.helloHttp = (req, res) => {
  res.send(`Hello ${escapeHtml(req.query.name || req.body.name ||
'World')}!`);
};
```

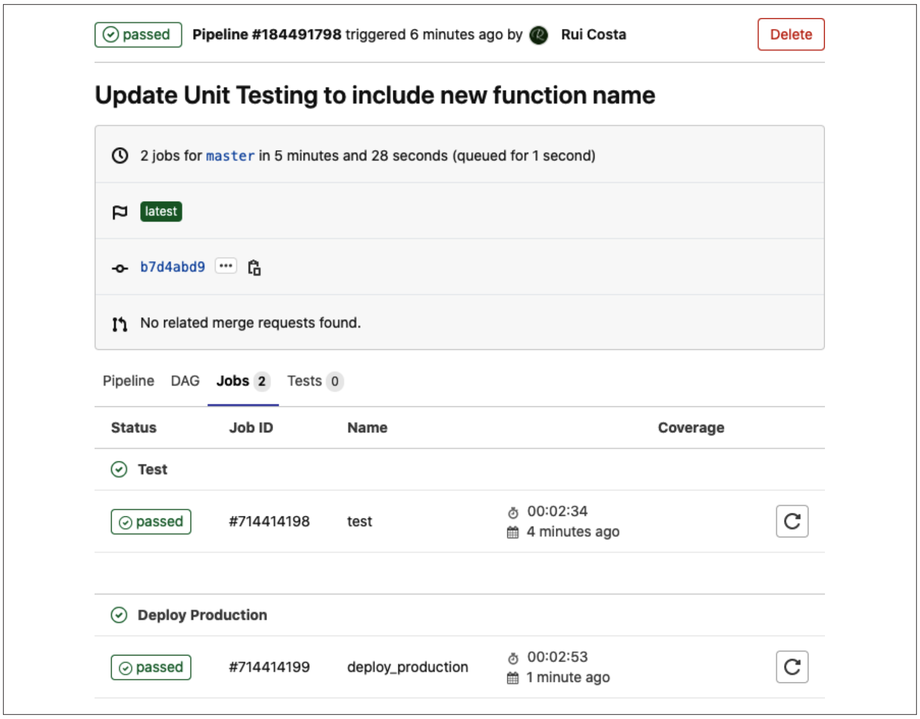
3. You will also need to install the mocha, sinon, and uuid npm packages in your *package.json* file. Your package file should be similar to the following:

```
{
  "name": "1-6-testing",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "test": "mocha test/index.test.js --exit"
```

```
      },
      "author": "Rui Santos Costa",
      "license": "ISC",
      "dependencies": {
        "escape-html": "^1.0.3"
      },
      "devDependencies": {
        "mocha": "^8.0.0",
        "sinon": "^9.0.0",
        "uuid": "^8.0.0"
      }
    }
```

For reference, you can read more in the documentation for Mocha and SINON.js.

4. Commit your changes and push the code to your repository. If you head back to GitLab CI/CD, you should see your job running.
5. You should a message once the pipeline has successfully executed (Figure 2-6).


passed Pipeline #184491798 triggered 6 minutes ago by Rui Costa Delete

Update Unit Testing to include new function name

2 jobs for master in 5 minutes and 28 seconds (queued for 1 second)

latest

b7d4abd9

No related merge requests found.

Pipeline DAG Jobs 2 Tests 0

Status	Job ID	Name		Coverage
Test				
passed	#714414198	test	00:02:34 4 minutes ago	
Deploy Production				
passed	#714414199	deploy_production	00:02:53 1 minute ago	

Figure 2-6. Completed GitLab job

6. Open the cloud function in the Google Cloud Console.
7. You should see the cloud function successfully deployed.

You have now successfully implemented unit testing by incorporating it into the GitLab pipeline and automating the deployment.

Discussion

In your repository, you created a file called *.gitlab-ci.yml*. The Stages section defines the stages for your GitLab pipeline. Here you have two stages: a test and `deploy_production`. On the test stage, the following actions will be performed at runtime. They must succeed before the next stage runs.

If one of these actions fails, the pipeline will fail, and your code will not be deployed:

- `apt update`
- `apt install -y nodejs npm`
- `npm install`
- `npm run test`

The npm run test is where you are telling GitLab at runtime to run the test script defined in *package.json*. In the sample code, the test script executes *index.test.js*:

```
"test": "mocha test/index.test.js --exit"
```

If the test fails, the GitLab pipeline will fail; however, if all tests pass, GitLab will deploy the cloud function.

2.8 Building an API Gateway to Gather Telemetry Data

Problem

You need a method to gather telemetry data for your API service running on cloud functions.

Solution

Build an Extensible Service Proxy v2 Beta (ESPv2 Beta) as an API gateway to collect telemetry data. The ESPv2 is an Envoy proxy that enables Cloud Endpoints to provide API management features.

1. Make a note of your project ID; the steps following ESP_PROJECT_ID are your project ID. To access your project ID, navigate to the Google Cloud Console and locate the Project Info card. It will list your project ID and project number.

2. Make a note of the project number; the steps following ESP_PROJECT_NUMBER are your project number.
3. The code is fairly long; please use the GitHub repository for this book: *https://github.com/ruiscosta/oreilly-google-cloud-cookbook*. Either clone or copy the code from the GitHub repository to your local workstation.
4. In the root directory of the code repository, deploy your cloud function:

   ```
   gcloud functions deploy hello-http-function --entry-point hello --
   runtime nodejs12 --trigger-http
   ```

5. Run the following command to deploy ESPv2 Beta to Cloud Run, change the `CLOUD_RUN_SERVICE_NAME` to the name you to want, and use the `ESP_PROJECT_ID` for the one you noted earlier:

   ```
   gcloud run deploy CLOUD_RUN_SERVICE_NAME \
   --image="gcr.io/endpoints-release/endpoints-runtime-serverless:2" \
   --allow-unauthenticated \
   --platform managed \
   --project=ESP_PROJECT_ID
   ```

6. When prompted for the region, choose us-central1.
7. On successful completion, the command displays a message similar to the following:

   ```
   Service [esphello] revision [esphello-00001-zuf] has been deployed and
   is serving 100 percent of traffic at https://esphello-6bc24kwh7a-
   uc.a.run.app
   ```

8. In this example, `https://esphello-6bc24kwh7a-uc.a.run.app` is the CLOUD_RUN_SERVICE_URL, and `esphello-6bc24kwh7a-uc.a.run.app` is the CLOUD_RUN_HOSTNAME.
9. Make a note of CLOUD_RUN_HOSTNAME. You will specify CLOUD_RUN_HOSTNAME in the host field of your *OpenAPI yaml* file.
10. You can verify that the initial version of ESPv2 Beta is deployed on Cloud Run by visiting the CLOUD_RUN_SERVICE_URL in your web browser. You should see a warning message about a missing environment variable. This warning message is expected.
11. Open the *openapi-functions.yaml* file located in the root directory of the code repository, using your favorite IDE.
12. In the Address field in the `x-google-backend` section, replace REGION with the Google Cloud region where your function is, FUNCTIONS_PROJECT_ID with your Google Cloud project ID, and FUNCTIONS_NAME with your function name. Here's an example:

    ```
    x-google-backend:
      address: https://us-central1-project.cloudfunctions.net/hello
      protocol: h2
    ```

13. In the `host` field, specify CLOUD_RUN_HOSTNAME, the hostname portion of the URL that Cloud Run created when you deployed ESPv2 Beta. Here's an example:

    ```
    info:
      title: Cloud Endpoints + GCF
      description: Cloud Endpoints with a Google Cloud Functions
      version: 1.0.0
    host: esphello-6bc24kwh7a-uc.a.run.app
    ```

14. To deploy the Endpoints configuration, run the following command and replace the ESP_PROJECT_ID with yours:

    ```
    gcloud endpoints services deploy openapi-functions.yaml \
        --project ESP_PROJECT_ID
    ```

15. Note the CONFIG_ID; we will need this later. For example, the CONFIG_ID for the following example is 2020-09-05-r3:

    ```
    Service Configuration [2020-09-05r3] uploaded for service
    [esphello-6bc24kwh7a-uc.a.run.app]
    ```

16. To enable your Endpoints service, run the following command:

    ```
    gcloud services enable ENDPOINTS_SERVICE_NAME
    ```

17. To determine the ENDPOINTS_SERVICE_NAME, go to the Endpoints page in the Cloud Console.

18. To build the service config into a new ESPv2 Beta docker image, run the following command:

    ```
    chmod +x gcloud_build_image \
    ./gcloud_build_image -s CLOUD_RUN_HOSTNAME \
        -c CONFIG_ID -p ESP_PROJECT_ID
         chmod +x gcloud_build_image
         ./gcloud_build_image -s esphello-6bc24kwh7a-uc.a.run.app \
          -c 2020-09-05r3 -p ruicosta-blog
    ```

19. The output should look like this:

    ```
    blog/endpoints-runtime-serverless:2.17.0-esphello-6bc24kwh7a-
    uc.a.run.app-2020-09-05r3
    ```

20. Keep a note of the full URI after "serverless:"; you will use that as your ESP_VERSION-CLOUD_RUN_HOSTNAME-CONFIG_ID in the next command.

21. Redeploy the ESPv2 Beta Cloud Run service with the new image. Replace CLOUD_RUN_SERVICE_NAME with the same Cloud Run service name you used when you originally deployed it in Step 5:

    ```
    gcloud run deploy esphello-6bc24kwh7a-uc.a.run.app \
    --image="gcr.io/ESP_PROJECT_ID/endpoints-runtime-serverless:ESP_VERSION-
    CLOUD_RUN_HOSTNAME-CONFIG_ID" \
    --allow-unauthenticated \
    ```

```
    --platform managed \
    --project=ESP_PROJECT_ID
```

22. Here's an example:

```
    gcloud run deploy esphello \
    --image="gcr.io/ruicosta-blog/endpoints-runtime-serverless:2.17.0-
    esphello-6bc24kwh7a-uc.a.run.app-2020-09-05r3" \
    --allow-unauthenticated \
    --platform managed \
    --project=ruicosta-blog
```

23. To test your deployed cloud function with Cloud Endpoints, run the following command:

```
    curl --request GET \
       --header "content-type:application/json" \
       "https://YOUR_ENDPOINT_HOST/hello"
```

24. You can view telemetry data for your function. In Endpoints, choose the Services page in the Google Cloud Console. Figures 2-7, 2-8, 2-9, and 2-10 provide an overview of some telemetry data you will collect for your API.

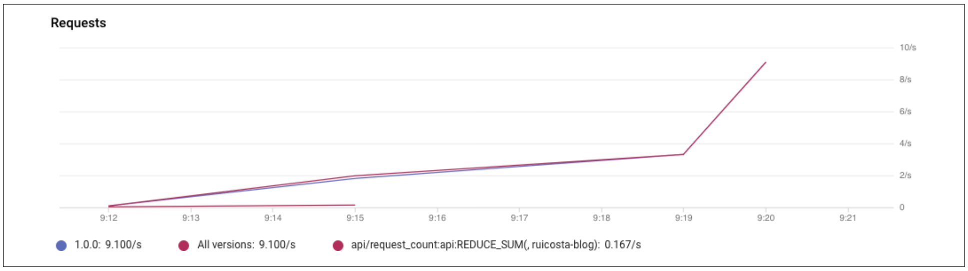


Figure 2-7. Endpoint requests

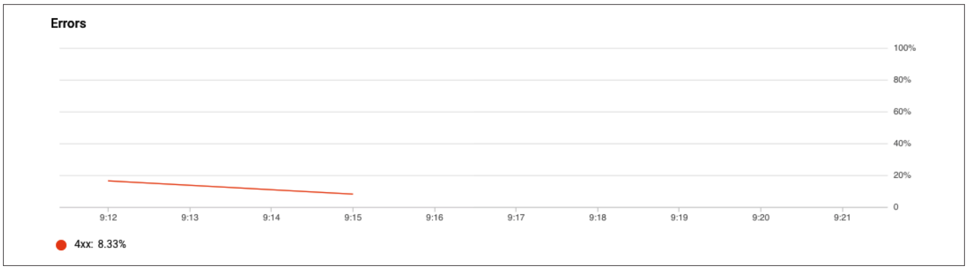


Figure 2-8. Endpoint errors

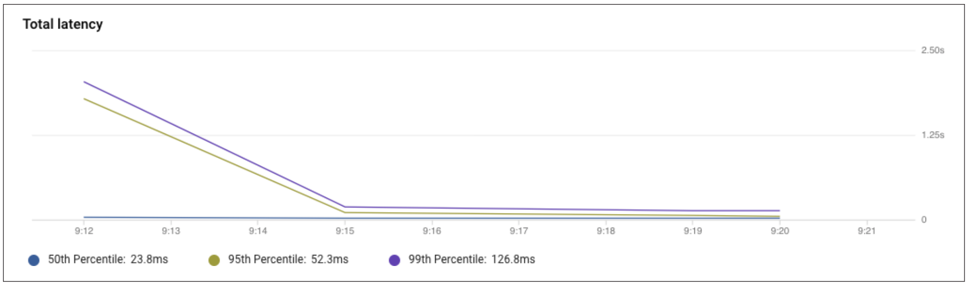
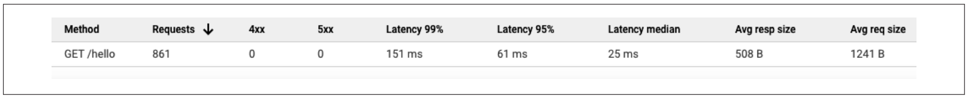


Figure 2-9. Endpoint latency



Method	Requests ↓	4xx	5xx	Latency 99%	Latency 95%	Latency median	Avg resp size	Avg req size
GET /hello	861	0	0	151 ms	61 ms	25 ms	508 B	1241 B

Figure 2-10. Endpoint summary

For additional information, review the following documentation:

- Google Cloud: Migration to the Extensible Service Proxy
- Envoy: Latest Updates
- Google Cloud: Cloud Endpoints

Discussion

In this recipe, you deployed Cloud Endpoints to intercept all requests to your function before invoking the function. When the function responds, Cloud Endpoints gathers and logs the telemetry data. This data is then available to you for reporting on the metrics collected.

CHAPTER 3

Google Cloud Run

Google Cloud Run is a serverless compute solution that allows you to run your containerized applications without having to manage the underlying infrastructure. Cloud Run is built on the same foundation as the open source community project Knative. Knative allows developers to focus on their application code without having to worry about the management of computing resources. Google Cloud Run also provides features such as autoscaling, redundancy, logging, and custom domains that enterprises require to run production workloads.

Google Cloud provides you with two ways to run Cloud Run: Cloud Run (fully managed) and Cloud Run for Anthos, which provides a consistent platform for your applications for either cloud or on-premises environments. Cloud Run for Anthos, which is powered by Knative, enables you to deploy your applications running on Cloud Run to platforms managed by Anthos.

Cloud Run (fully managed) allows you to deploy containers without having to worry about the underlying infrastructure. Cloud Run for Anthos allows you to run your containers on-premises or on a Google Cloud Kubernetes Engine (GKE) cluster. Running on a GKE cluster does require the management of the cluster but provides additional benefits such as custom machine types and additional networking support. This chapter, however, will focus on Cloud Run fully managed.

In this chapter, you will learn how to trigger a Cloud Run service from a Pub/Sub topic creating automated pipelines, use your custom domain to increase your branding efforts by using a custom domain, and learn how to run blue/green deployments to release your application by directing traffic between two Cloud Run services running different versions of an application.

Google Cloud Run can be written in any programming language of your choice, but in this chapter you will be using Go. All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each individual recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.
4. Enabled the Cloud Functions and Cloud Build APIs:

```
gcloud services enable run.googleapis.com
gcloud services enable cloudbuild.googleapis.com
gcloud services enable \
    containerregistry.googleapis.com
```

3.1 Deploying a Prebuilt Hello World Container

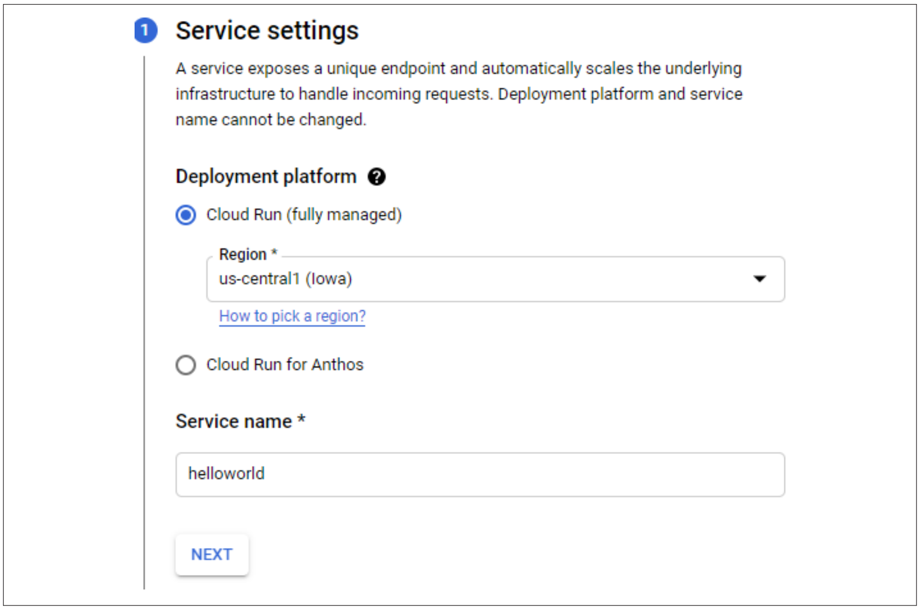
Problem

You want to deploy your first Cloud Run prebuilt container application to perform a simple Hello World.

Solution

Leverage an existing container image running on Google Cloud Source Repository located at *gcr.io/cloudrun/hello* to deploy a new Cloud Run container that responds to incoming web requests.

1. Open Cloud Run in the Google Cloud Console.
2. Click Create Service.
3. Select a region and enter a name. Figure 3-1 shows the service name is hello-world.



Service settings

A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Deployment platform and service name cannot be changed.

Deployment platform

● Cloud Run (fully managed)

Region *
us-central1 (Iowa)

How to pick a region?

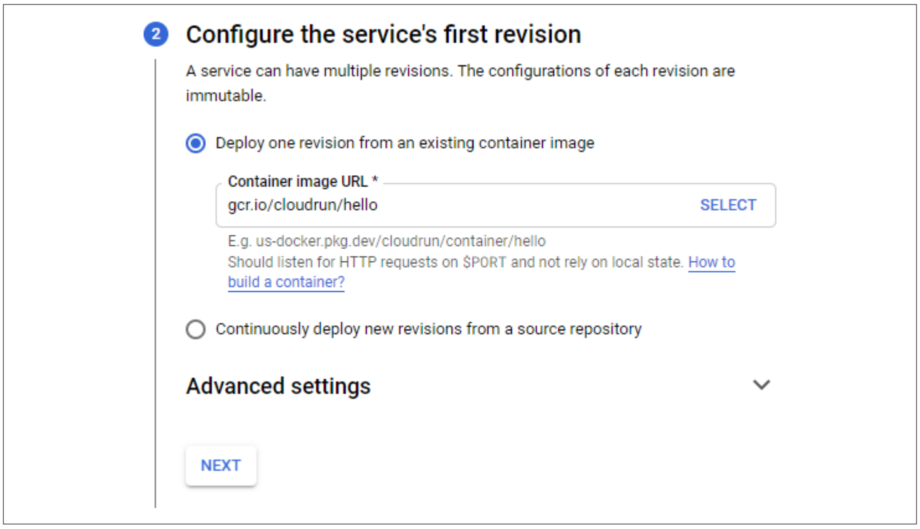
○ Cloud Run for Anthos

Service name *

helloworld

Figure 3-1. Cloud Run Service Settings

4. Click Next.
5. For the container image URL, enter **`gcr.io/cloudrun/hello`**, as shown in Figure 3-2.



Configure the service's first revision

A service can have multiple revisions. The configurations of each revision are immutable.

● Deploy one revision from an existing container image

Container image URL *
gcr.io/cloudrun/hello

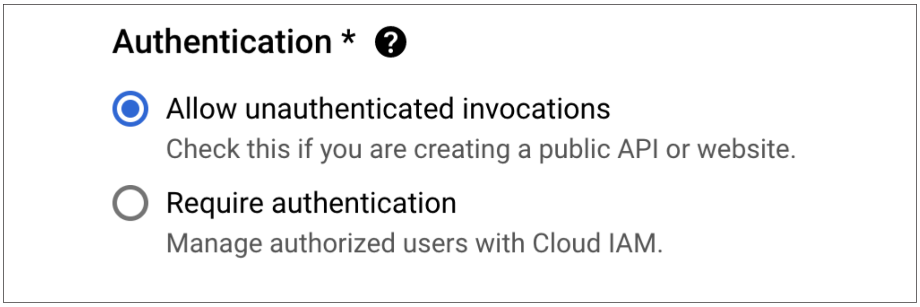
E.g. us-docker.pkg.dev/cloudrun/container/hello
Should listen for HTTP requests on $PORT and not rely on local state. How to build a container?

○ Continuously deploy new revisions from a source repository

Advanced settings

Figure 3-2. Cloud Run service version

6. Click Next.
7. For authentication, select Allow Unauthenticated Invocations, as shown in Figure 3-3.



Authentication * ?

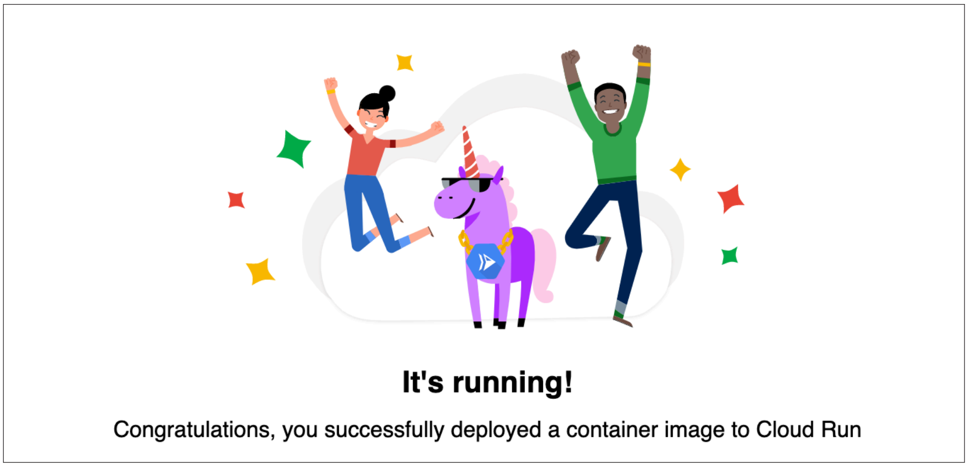
[✓ Allow unauthenticated invocations]
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

Figure 3-3. Cloud Run Authentication

8. Click Create.
9. Click the URL displayed to launch your newly deploy Cloud Run container.

You should see the following page in your browser page, shown in Figure 3-4.



It's running!

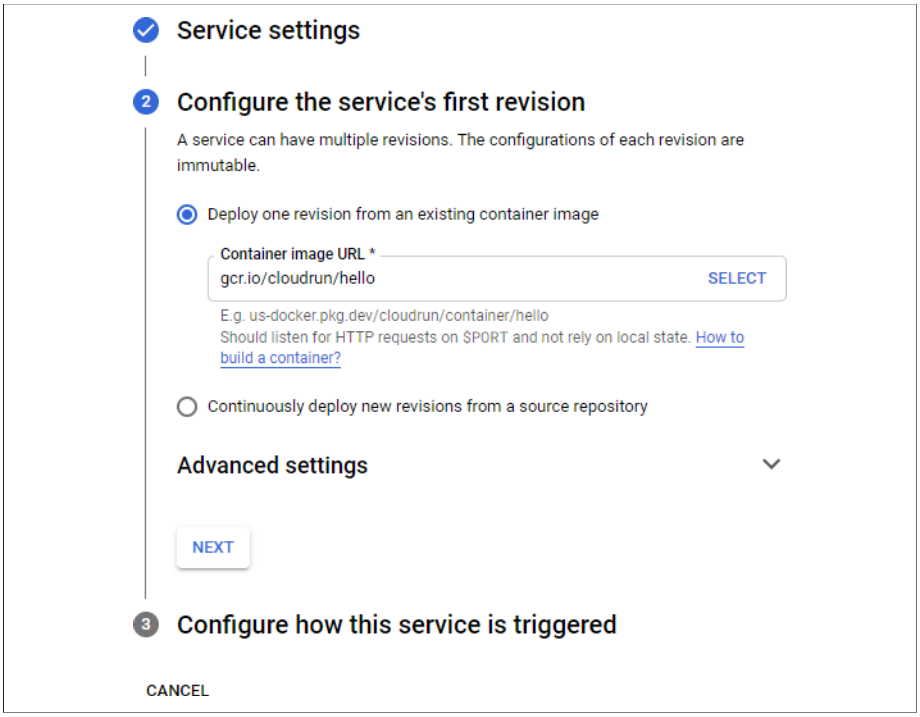
Congratulations, you successfully deployed a container image to Cloud Run

Figure 3-4. Cloud Run service successful

Discussion

This is a simple Hello World application, but it demonstrates some of the basic concepts in deploying Cloud Run services, such as:

- Using an existing public container referenced as gcr.io, as shown in Figure 3-5.


Service settings

Configure the service's first revision

A service can have multiple revisions. The configurations of each revision are immutable.

(●) Deploy one revision from an existing container image

Container image URL *
gcr.io/cloudrun/hello

E.g. us-docker.pkg.dev/cloudrun/container/hello
Should listen for HTTP requests on $PORT and not rely on local state. How to build a container?

( ) Continuously deploy new revisions from a source repository

Figure 3-5. Cloud Run Service Settings

- Allowing unauthenticated invocations, which give public anonymous users access to your application.
- Using a fully managed environment to abstract all the infrastructure so you can run your application quickly.

3.2 Building Your Own Hello World Container

Problem

You want to create a container for your application and deploy it to Google Cloud Run.

Solution

Build a container with your application code, deploy it to Google Cloud Container Registry, and finally deploy your newly deployed container to Cloud Run.

1. Create a new directory on your local machine called *helloworld*.
2. Create a *go.mod* file and enter the following code to it:

```
module github.com/GoogleCloudPlatform/golang-samples/run/helloworld
go 1.15
```

A *go.mod* file defines the module's path and its dependency requirements.

3. Create a new file named *main.go* and type the following code into it:

```
package main

import (
        "fmt"
        "log"
        "net/http"
        "os"
)

func main() {
        log.Print("starting server...")
        http.HandleFunc("/", handler)

        // Determine port for HTTP service.
        port := os.Getenv("PORT")
        if port == "" {
                port = "8080"
                log.Printf("defaulting to port %s", port)
        }

        // Start HTTP server.
        log.Printf("listening on port %s", port)
        if err := http.ListenAndServe(":"+port, nil); err != nil {
                log.Fatal(err)
        }
}

func handler(w http.ResponseWriter, r *http.Request) {
        name := os.Getenv("NAME")
        if name == "" {
                name = "World"
        }
        fmt.Fprintf(w, "Hello %s!\n", name)
}
```

4. To containerize the application, create a new file named *Dockerfile* in the *hello-world* directory, and type the following code:

   ```
   FROM golang:1.15-buster as builder
   WORKDIR /app
   COPY go.* ./
   RUN go mod download
   COPY . ./
   RUN go build -mod=readonly -v -o server
   FROM debian:buster-slim
   RUN set -x && apt-get update && DEBIAN_FRONTEND=noninteractive apt-get
   install -y \
       ca-certificates && \
       rm -rf /var/lib/apt/lists/*
   COPY --from=builder /app/server /app/server
   CMD ["/app/server"]
   ```

5. Create a *.dockerignore* file to exclude files from your container:

   ```
   # Exclude locally vendored dependencies.
   vendor/
   # Exclude "build-time" ignore files.
   .dockerignore
   .gcloudignore

   # Exclude git history and configuration.
   .gitignore
   ```

6. Build your container with the sample application image, using Cloud Build. Run the following command from the directory containing the Dockerfile and replace `PROJECT-ID` with your Google Cloud Project ID:

   ```
   gcloud builds submit --tag gcr.io/PROJECT-ID/helloworld
   ```

7. Deploy the Cloud Run service, using the newly deployed container image by running the following command:

   ```
   gcloud run deploy --image gcr.io/PROJECT-ID/helloworld --platform
   managed
   ```

8. Press Enter to accept the default name, helloworld.
9. Select a Google Cloud region, for example, us-central1.
10. You will be prompted to allow unauthenticated invocations. Enter **y**.

Once the command is completed successfully, it will display the Cloud Run service URL. You can click or copy the URL and open it in a browser.

Discussion

You have successfully created a container image, deployed it to Google Cloud Container Registry, and created a Cloud Run service from the deployed container image. In this recipe, you also learned how to deploy a Cloud Run service with the gcloud command versus using the Google Cloud Console interface. You also learned how to build your own container, deploy it to Container Registry, and run the application on Cloud Run.

3.3 Using Cloud Run with a Custom Domain

Problem

You don't want to use the generated URL that Google Cloud provides for your Google Cloud service and prefer to use your own domain.

Solution

Use your own domain to map a domain or subdomain to the cloud Hello World created in Recipe 3.1.

1. Verify ownership of your domain. To list the currently verified domains, run the following command:

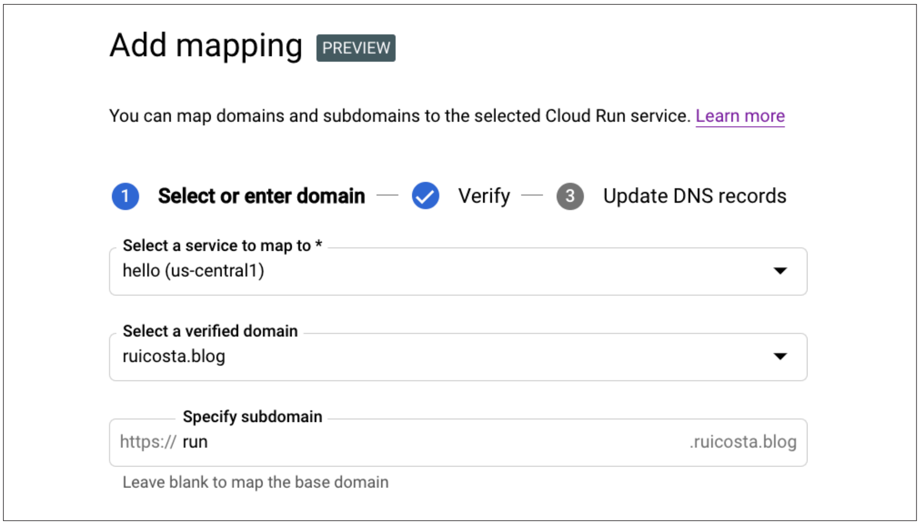
   ```
   gcloud domains list-user-verified
   ```

2. If the command does not list any output, you do not have any verified domains.
3. Run the following command to verify your domain:

   ```
   gcloud domains verify BASE-DOMAIN
   Example: gcloud domains verify ruicosta.blog
   ```

> For example, if you want to map, say, run.ruicosta.blog, you will need to verify the root domain, which is ruicosta.blog.

4. The command will open a new browser window that will take you to Google Webmaster Central. Follow the instructions to verify your domain.
5. Open the MANAGE CUSTOM DOMAINS page on the Cloud Run services page within Google Cloud Console.
6. Click ADD MAPPING.
7. Choose your service to map to.
8. Select your verified domain.
9. Optionally: Choose a subdomain. You can choose to map to the root of the verified domain or to a subdomain, as shown in Figure 3-6.



Add mapping PREVIEW

You can map domains and subdomains to the selected Cloud Run service. Learn more

1 **Select or enter domain** — ✓ Verify — 3 Update DNS records

Select a service to map to *: hello (us-central1)

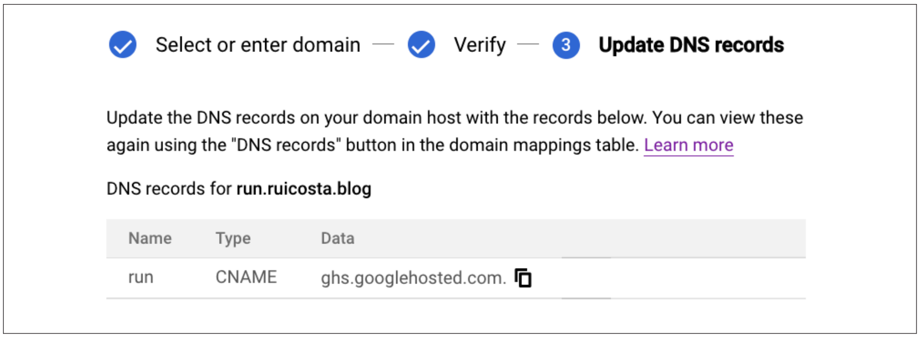
Select a verified domain: ruicosta.blog

Specify subdomain: https:// run .ruicosta.blog

Leave blank to map the base domain

Figure 3-6. Cloud Run domain mapping

10. Click Continue.
11. Update your DNS records at your domain registrar, such as GoDaddy, Namecheap, or Google Domains, using the DNS records displayed in the previous step. Figure 3-7 shows the information required to register the associated CNAME record in your registrar.



✓ Select or enter domain — ✓ Verify — 3 **Update DNS records**

Update the DNS records on your domain host with the records below. You can view these again using the "DNS records" button in the domain mappings table. Learn more

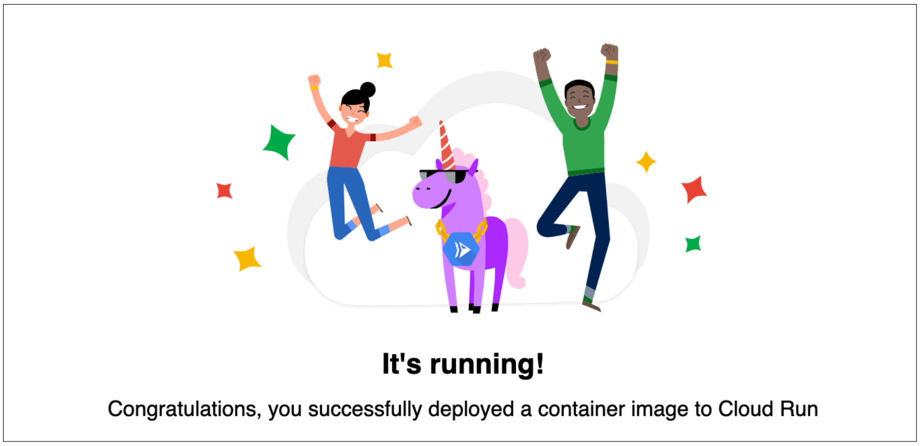
DNS records for **run.ruicosta.blog**

Name	Type	Data
run	CNAME	ghs.googlehosted.com.

Figure 3-7. Cloud Run domain mapping

12. Test your new custom domain mapping by going to your mapped domain in your web browser, as shown in Figure 3-8.

    Please note for your custom domain mapping that it might take a few minutes for it to be updated with the newly verified domain.


It's running!

Congratulations, you successfully deployed a container image to Cloud Run

Figure 3-8. Successful Cloud Run domain mapping

Discussion

With custom domain mapping, you can direct users to your Cloud Run service running with your domain name versus using the Google Cloud domain name. An easy-to-read (and easy-to-spell) domain name enhances your branding efforts. Please visit the Google Cloud Run documentation, because custom domain names are not supported in all regions; check whether your region supports custom domains.

3.4 Triggering a Cloud Run from Cloud Pub/Sub

Problem

You want to trigger an operation when a message is published to a cloud Pub/Sub topic. For example, a new message is received from your application by Cloud Pub/Sub and you want to execute an operation on the arrived message.

Solution

Set up Cloud Run to listen to arriving messages in Cloud Pub/Sub. When a new message arrives, Cloud Run can run an operation on the newly arrived message.

1. Create a new Pub/Sub topic by running the following command:

```
gcloud pubsub topics create chapter-3-4-topic
```

2. Create a new directory called *chapter-3-4* on your local machine.
3. Create a *go.mod* file and enter the following code to it:

```
module github.com/GoogleCloudPlatform/golang-samples/run/pubsub
go 1.15
```

> A *go.mod* file defines the module's path and its dependency requirements.

4. Create a new file named *main.go* and type the following code into it:

```
package main
import (
    "encoding/json"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    http.HandleFunc("/", HelloPubSub)
    // Determine port for HTTP service.
    port := os.Getenv("PORT")
    if port == "" {
         port = "8080"
         log.Printf("Defaulting to port %s", port)
    }
    // Start HTTP server.
    log.Printf("Listening on port %s", port)
    if err := http.ListenAndServe(":"+port, nil); err != nil {
         log.Fatal(err)
    }
}

type PubSubMessage struct {
    Message struct {
         Data []byte `json:"data,omitempty"`
         ID   string `json:"id"`
    } `json:"message"`
    Subscription string `json:"subscription"`
}

func HelloPubSub(w http.ResponseWriter, r *http.Request) {
    var m PubSubMessage
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
            log.Printf("ioutil.ReadAll: %v", err)
            http.Error(w, "Bad Request", http.StatusBadRequest)
            return
        }
        if err := json.Unmarshal(body, &m); err != nil {
            log.Printf("json.Unmarshal: %v", err)
            http.Error(w, "Bad Request", http.StatusBadRequest)
            return
        }

        name := string(m.Message.Data)
        if name == "" {
            name = "World"
        }
        log.Printf("Hello %s!", name)
    }
```

You return a code from the service such as HTTP 200 or 204, which acknowledges complete processing of the Pub/Sub message. Error codes, such as HTTP 400 or 500, indicate the message will be retried.

5. To containerize the application, create a new file named *Dockerfile* in *chapter-3-4*, and enter the following code:

```
FROM golang:1.15-buster as builder

# Create and change to the app directory.
WORKDIR /app

# Retrieve application dependencies.
COPY go.* ./
RUN go mod download

# Copy local code to the container image.
COPY . ./

# Build the binary.
RUN go build -mod=readonly -v -o server

FROM debian:buster-slim
RUN set -x && apt-get update && DEBIAN_FRONTEND=noninteractive apt-get
install -y \
    ca-certificates && \
    rm -rf /var/lib/apt/lists/*

# Copy the binary to the production image from the builder stage.
COPY --from=builder /app/server /server
```

```
# Run the web service on container startup.
CMD ["/server"]
```

6. Build the container image and push it to Google Cloud Container Registry:

   ```
   gcloud builds submit --tag gcr.io/PROJECT_ID/pubsub
   ```

7. Deploy the Cloud Run service from the newly deployed container image and replace PROJECT_ID with your cloud project ID:

   ```
   gcloud run deploy pubsub --image gcr.io/PROJECT_ID/pubsub --platform
   managed
   ```

   a. Press Enter to accept the default name, pubsub.

   b. Select the region of your choice, for example, us-central1.

   c. You will be prompted to allow unauthenticated invocations. Enter **N**. You will provide Pub/Sub the invoker service account permission so it can invoke the Cloud Run service.

8. Enable the Pub/Sub service to generate authentication tokens in your project:

   ```
   gcloud projects add-iam-policy-binding PROJECT_ID \   --
   member=serviceAccount:service
   -PROJECT-NUMBER@gcp-sa-pubsub.iam.gserviceaccount.com \
   --role=roles/iam.serviceAccountTokenCreator
   ```

   a. Replace PROJECT_ID with your Google Cloud project ID.

   b. Replace PROJECT_NUMBER with your Google Cloud project number.

9. Create a service account for the Pub/Sub subscription to use:

   ```
   gcloud iam service-accounts create cloud-run-pubsub-invoker \
        --display-name "Cloud Run Pub/Sub Invoker"
   ```

10. Give the newly created service account permission to invoke your Pub/Sub Cloud Run service:

    ```
    gcloud run services add-iam-policy-binding pubsub \
    --member=serviceAccount:cloud-run-pubsub-
    invoker@PROJECT_ID.iam.gserviceaccount.com \
       --role=roles/run.invoker
    ```

    Replace PROJECT_ID with your Google Cloud project ID.

11. Create a Pub/Sub subscription with the service account created in step 9:

    ```
    gcloud pubsub subscriptions create myRunSubscription --topic
    chapter-3-4-topic \
    --push-endpoint=SERVICE-URL/ \
    --push-auth-service-account=cloud-run-pubsub-
    invoker@PROJECT_ID.iam.gserviceaccount.com
    ```

    a. Replace the SERVICE-URL with the Cloud Run service URL.

    b. Replace the PROJECT_ID with your Google Cloud project ID.

12. To test the newly deployed Cloud Run service, we will trigger it with the newly deployed Cloud Pub/Sub subscription:

    ```
    gcloud pubsub topics publish chapter-3-4-topic --message "Test Runner"
    ```

13. Navigate to the Cloud Run service logs:
    a. Click the pubsub service.
    b. Select the Logs tab.
    c. Look for the “Hello Test Runner!” message.

Discussion

By leveraging Cloud Pub/Sub, you can invoke a Cloud Run service to perform the specified operation. Google Cloud Pub/Sub is a messaging service for exchanging message data among applications and services. By abstracting the senders and receivers, it allows for multiple applications written in different languages to communicate with each other. This process allows you to trigger an operation only when Cloud Pub/Sub receives a message. This becomes a powerful feature to perform background operations when certain events occur.

In the following example, as shown in Figure 3-9, you send an SMS when the user subscribes to your service. This works by using Cloud Pub/Sub as your middleware, which accepts a message payload from the user and then, using a Cloud Pub/Sub subscriber, you collect the message payload and perform an operation on this payload. In this example, your operation is to send an SMS to the user, confirming that they have subscribed to your service.

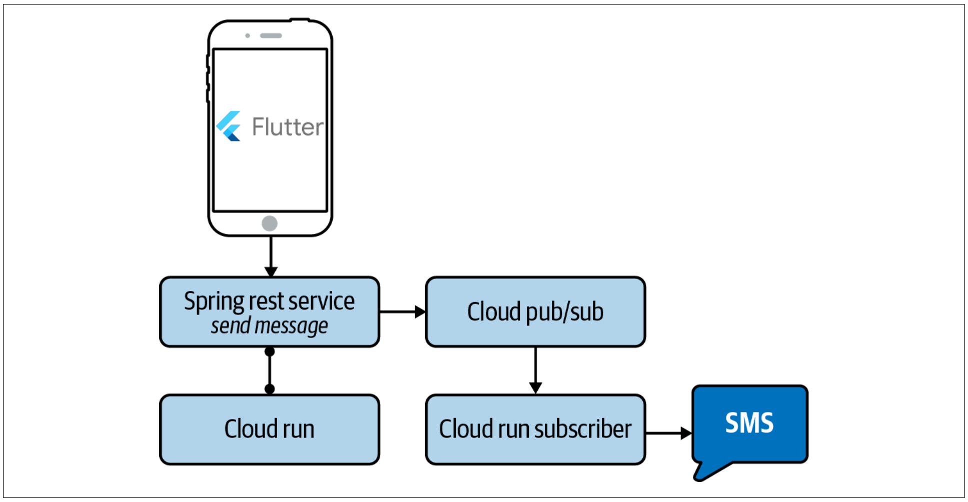


Figure 3-9. Cloud Run and Pub/Sub flow

3.5 Deploying a Web Application to Cloud Run

Problem

You need a method to deploy a static web application to Cloud Run.

Solution

Create a Cloud Run service that uses an NGINX container to serve your web files.

1. Create a new directory called *cloudrun* on your local machine.
2. Create the following directories in the root directory of cloudrun: *html* and *nginx*.
3. Go to the HTML directory:

   ```
   cd html
   ```

4. Create a new file named *index.html* in the HTML directory and type the following code into it:

   ```
   <html>
   <body>Hello from Google Cloud Run and NGINX</body>
   </html>
   ```

5. Go to the *nginx* directory.

   ```
   cd nginx
   ```

6. Create a new file named *default.conf* in the *nginx* directory and type the following code into it:

   ```
   server {
       listen       8080;
       server_name  localhost;
       location / {
           root   /usr/share/nginx/html;
           index  index.html index.htm;
       }
       # redirect server error pages to the static page /50x.html
       error_page   500 502 503 504  /50x.html;
       location = /50x.html {
           root   /usr/share/nginx/html;
       }
   }
   ```

7. Go to the root *cloudrun* directory.

8. To containerize the application, create a new file named *Dockerfile* in the *cloudrun* directory created in step 1, and enter the following code:

```
FROM nginx
COPY html /usr/share/nginx/html
COPY nginx/default.conf /etc/nginx/conf.d/default.conf
```

In this Dockerfile, we are creating a new container based on the NGINX container, adding the static HTML file as well as setting the NGINX configuration file.

9. Build your container by running the following command from the root of the *cloudrun* directory:

```
gcloud builds submit --tag gcr.io/PROJECT-ID/nginx-web
```

Replace `PROJECT-ID` with your Google Cloud Project ID.

10. Deploy the Cloud Run service, using the newly deployed container image, by running the following command:

```
gcloud run deploy --image gcr.io/PROJECT-ID/nginx-web --platform managed
```

   a. Press Enter to accept the default name, nginx-web.
   b. Select the region of your choice, for example, us-central1.
   c. You will be prompted to allow unauthenticated invocations. Enter **Y**.

11. Once the command is completed successfully, it will display the Cloud Run service URL. You can click or copy the URL and open it in a browser. You should now see the newly deployed container serving your HTML content with NGINX, as shown in Figure 3-10.

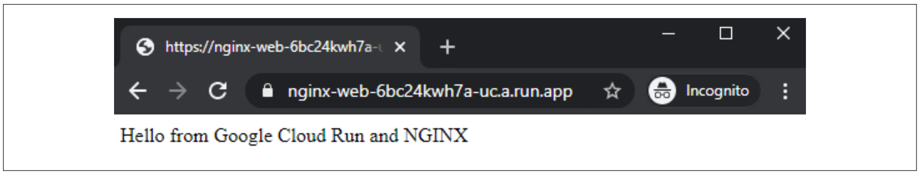

https://nginx-web-6bc24kwh7a-u
nginx-web-6bc24kwh7a-uc.a.run.app
Incognito
Hello from Google Cloud Run and NGINX

Figure 3-10. Cloud Run running NGINX

Discussion

Google Cloud provides other services to serve static and dynamic websites. You can use Google Cloud Storage to run static content or even a content delivery network (CDN), but if you need to deploy dynamic content, you will need to host your application on a computing platform. For small static websites, plus dynamic content, you can use Cloud Run, since it provides customization if needed to the underlying runtime, and it allows you to serve static content as well.

3.6 Rolling Back a Cloud Run Service Deployment

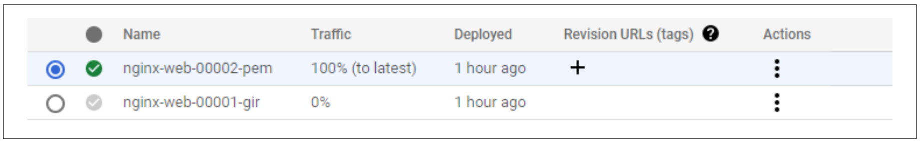
Problem

You need to roll back your Cloud Run service deployment due to a bug you found in your code.

Solution

Cloud Run allows you to deploy multiple revisions of your application. With this feature, you can roll back to a previous revision.

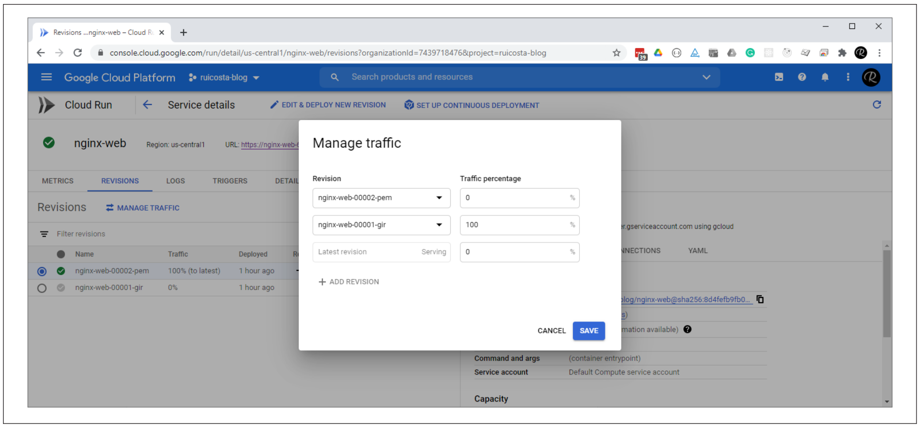
1. To roll back to a previous revision, go to the Google Cloud Console and navigate to Cloud Run.
2. Locate the service you need to roll back.
3. The revisions tab will list all the available versions of your service, as shown as Figure 3-11.



	●	Name	Traffic	Deployed	Revision URLs (tags)	Actions
◉	✓	nginx-web-00002-pem	100% (to latest)	1 hour ago	+	⋮
○	✓	nginx-web-00001-gir	0%	1 hour ago		⋮

Figure 3-11. Cloud Run revisions

4. In the list of revisions, select the revision you are rolling back.
5. Click Manage Traffic, as shown in Figure 3-12.
   a. Select the revision you want to roll back to from the drop-down list.
   b. Set that previous revision's traffic percentage to 100.
   c. Set the currently serving revision's percentage to 0.
   d. Click Save.



Manage traffic

Revision	Traffic percentage
nginx-web-00002-pem	0 %
nginx-web-00001-gir	100 %
Latest revision (Serving)	0 %

Name	Traffic	Deployed
nginx-web-00002-pem	100% (to latest)	1 hour ago
nginx-web-00001-gir	0%	1 hour ago

Figure 3-12. Cloud Run Manage Traffic

6. Once you click Save, the traffic will be transitioned to the version you selected, allowing you to roll back to older versions.

Discussion

The ability to roll back to previous Cloud Run versions enables you to fix errors quickly in the current version being served.

3.7 Deploying Cloud Run Services in a Gradual Rollout

Problem

You need to deploy a Cloud Run service, using the blue/green deployment methodology. This allows you to release your application by directing traffic between two Cloud Run services running different versions of an application.

Solution

Cloud Run allows you to split traffic between multiple versions of your Cloud Run service so you can roll out blue/green deployments.

1. To roll out a new Cloud Run service version gradually, navigate to the Cloud Run services in the Google Cloud Console.
2. Locate the service you want to deploy a new revision gradually to.
3. Click Deploy New Revision.
4. Fill out the parameters as needed; make sure the checkbox labeled Serve This Revision Immediately is cleared.

5. Click Deploy.
6. Click Manage Traffic. The new revision is currently not serving any traffic, because you cleared the Service This Revision Immediately checkbox.
7. On the Manage Traffic page:
   a. Set it to the desired percentage for the new revision, for example, 10.
   b. Click Save.
   c. Repeat step 7 but change the percentage value to say 20, increasing the percentage as needed for the new revision.

Discussion

With the ability to roll out a blue/green deployment gradually, you can validate that your updated revision is working as expected. If something occurs with the new revision, only the percentage of the served traffic would be affected. If you need to, you can roll back 100% of the traffic, which was shown in Recipe 3.6.

3.8 Cloud Run Configuration Parameters

Problem

You need to fine-tune Cloud Run parameters, such as setting how many maximum instances are running, setting CPU allocation, and adjusting the request timeout.

Solution

Cloud Run provides many options for tuning your service, including setting request timeouts and how many instances to run, and even updating CPU allocation, all through the Google Cloud Console.

CPU allocation

Cloud Run by default allocates one CPU for each container instance. You can adjust the CPU allocation by using the Cloud Console.

1. Navigate to the Cloud Run services in the Google Cloud Console.
2. Click the service you want to adjust the CPU allocation for and click Edit And Deploy New Revision.

3. In Advanced Settings, click Container.
4. Select the desired CPU allocation from the drop-down list.
5. Click Deploy.

Request timeout

The request timeout specifies the time by which Cloud Run must return a response. If a response isn't returned within the time specified, the request returns an Error 504. The default timeout response for Cloud Run is five minutes.

1. Navigate to the Cloud Run services in the Google Cloud Console.
2. Click the service you want to adjust the request timeout for and click Edit And Deploy New Revision.
3. In Advanced Settings, click Container.
4. In the Request Timeout field, enter the timeout value that you want to use in seconds.
5. Click Deploy.

Maximum number of instances

The maximum number of instances in Cloud Run allows you to limit the number of instances that will be enabled to service incoming requests. You can use this setting to control your costs or control connections to, say, backend services due to limitations on scaling on your backend services.

1. Navigate to the Cloud Run services in the Google Cloud Console.
2. Click the service you want to adjust the maximum number of instances for and click Edit And Deploy New Revision.
3. In Advanced Settings, click Container.
4. Enter the maximum number of instances. You can use any value from 1 to 1,000.
5. Click Deploy.

Minimum number of instances

Cloud Run scales to X number of instances based on the number of incoming requests to the service. If your service requires reduced latency due to cold starts, you can set the minimum number of container instances to be ready to serve requests and avoid the issue of having to start new instances referred to as cold starts.

1. Navigate to the Cloud Run services in the Google Cloud Console.
2. Click the service you want to adjust the minimum number of instances for and click Edit And Deploy New Revision.
3. In Advanced Settings, click Container.
4. Enter a value for the minimum number of instances. This will allow the value of instances listed to be kept warm and ready to receive requests.
5. Click Deploy.

CHAPTER 4
Google App Engine

Google App Engine is a serverless compute solution that allows you to run your applications without having to manage the underlying infrastructure. App Engine supports a variety of programming languages including Node.js, Java, Ruby, C#, Go, Python, and PHP; you can even use a unsupported language by using containers. App Engine has two editions: Standard and Flexible. Flexible allows you to bring any library and framework to App Engine.

App Engine provides you with enterprise-ready deployment features such as application versioning, traffic splitting, security, monitoring, and debugging. With App Engine, all you need to focus on is your code; Google Cloud manages the underlying infrastructure. In this chapter, you will learn how to deploy your application with a CI/CD pipeline, secure it, map a custom domain, use ML APIs, and debug your application.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.
4. Enabled the Cloud Functions and Cloud Build APIs.

```
gcloud services enable cloudbuild.googleapis.com
gcloud services enable \
   containerregistry.googleapis.com
```

4.1 Deploying a Hello World to App Engine (Standard)

Problem

You want to deploy your first App Engine application to perform a simple Hello World.

Solution

Use the Google Cloud command line and your favorite editor to build a simple Express.js application to run on App Engine.

1. On your local workstation, create a temporary folder to hold the files you will need to create the Hello World application.
2. In your favorite IDE, create an *app.js* file in the root of the directory you created in step 1 and copy the following code to the file:

    ```
    'use strict';
    const express = require('express');
    const app = express();
    app.get('/', (req, res) => {
        res.status(200).send('Hello, world!').end();
    });
    const PORT = process.env.PORT || 8080;
    app.listen(PORT, () => {
        console.log(`App listening on port ${PORT}`);
        console.log('Press Ctrl+C to quit.');
    });
    module.exports = app;
    ```

3. Now create an *app.yaml* file in the root of the same directory and copy the following code to the file. The *app.yaml* file defines the settings for your application, including the runtime of your code.

    ```
    runtime: nodejs14
    ```

4. Now create a *package.json* file in the root of the same directory and copy the following code to the file:

    ```
    {
      "name": "appengine-hello-world",
      "engines": {
        "node": ">=14.0.0"
      },
      "scripts": {
        "start": "node app.js"
      },
      "dependencies": {
        "express": "^4.17.1"
    ```

```
        }
    }
```

5. In your terminal, run the following command in the root directory you created in step 1:

   ```
   npm install
   ```

6. To deploy the application to App Engine Standard, run the following command:

   ```
   gcloud app deploy
   ```

7. To view your deployed application, run the following command:

   ```
   gcloud app browse
   ```

This will open the application in your default browser.

Discussion

You have successfully deployed your first App Engine application using Node.js. It is a simple Hello World application but demonstrates some of the basic concepts in deploying App Engine services. To review the differences between App Engine Standard and Flexible, jump to Table 4-1 that lists the differences.

4.2 Deploying a Hello World to App Engine (Flexible)

Problem

You want to deploy an App Engine application running as a container to perform a simple Hello World.

Solution

App Engine Flexible supports running a Docker container that can include custom runtimes or other source code written in a different programming language. Since App Engine Flexible supports running Docker containers, you will use the Flexible version of App Engine to deploy a simple Hello World.

1. On your local workstation, create a temporary folder to hold the files you will need to create the Hello World application.
2. In your favorite IDE, create a *Dockerfile* in the root of the directory you created in step 1 and copy the following code to the file:

   ```
   FROM nginx
   COPY nginx.conf /etc/nginx/nginx.conf
   RUN mkdir -p /var/log/app_engine
   RUN mkdir -p /usr/share/nginx/www/_ah && \
       echo "healthy" > /usr/share/nginx/www/_ah/health
   ```

```
ADD www/ /usr/share/nginx/www/
RUN chmod -R a+r /usr/share/nginx/www
```

The FROM command builds a base image, using the official NGINX Docker image.

3. Create a new file named *app.yaml* in the root of your temporary directory and type the following code into it:

   ```
   runtime: custom
   env: flex
   ```

4. Now create a new file named *nginx.conf*, also in the root of your temporary directory you create, and type the following code into it:

   ```
   events {
       worker_connections 768;
   }

   http {
       sendfile on;
       tcp_nopush on;
       tcp_nodelay on;
       keepalive_timeout 65;
       types_hash_max_size 2048;
       include /etc/nginx/mime.types;
       default_type application/octet-stream;
       access_log /var/log/app_engine/app.log;
       error_log /var/log/app_engine/app.log;
       gzip on;
       gzip_disable "msie6";
       server {
           listen 8080;
           root /usr/share/nginx/www;
           index index.html index.htm;
       }
   }
   ```

5. Create a new folder called *www* in the root of your temporary directory.
6. Within the *www* folder, create a new file called *index.html* and copy the following code to it:

   ```
   <!doctype html>
   <html>
     <head>
       <title>Hello World!</title>
     </head>
     <body>
       <h1>Welcome to nginx!</h1>
   ```

```
      <p>Brought to you by Google App Engine.</p>
    </body>
  </html>
```

7. Run the following command to deploy your application to App Engine:

```
gcloud app deploy
```

8. To view your deployed application, run the following command:

```
gcloud app browse
```

This will open the application in your default browser.

Discussion

You have successfully deployed a static web application running on an NGINX web server as a custom runtime on App Engine Flexible. App Engine Flexible is a perfect choice for applications that:

- Need a custom runtime.
- Depend on frameworks that are not supported by App Engine Standard.

Table 4-1 summarizes the differences between App Engine Standard and Flexible at a high level.

Table 4-1. Differences Between App Engine Standard and Flexible

Feature	Standard environment	Flexible environment
Instance startup time	Seconds	Minutes
SSH debugging	No	Yes
Scaling	Manual, basic, automatic	Manual, automatic
Scale to zero	Yes	No, minimum 1 instance
Modifying the runtime	No	Yes (through Dockerfile)
Deployment time	Seconds	Minutes
WebSockets	No	Yes
Supports installing third-party binaries	Yes	Yes

4.3 Securing Your Application with Identity-Aware Proxy

Problem

You've deployed your Hello World application running on App Engine Flexible and want only certain users to be able to access it.

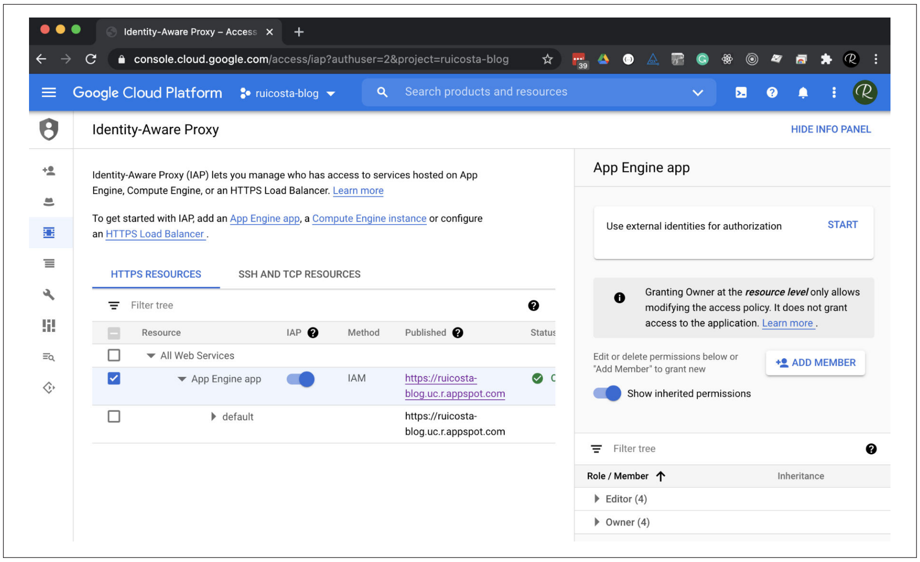
Solution

Use the Google Cloud Identity-Aware proxy (IAP) to restrict access to only a set of predefined users. IAP provides a single point of control for managing user access to your applications. We will use the Cloud Hello World application created in Recipe 4.1 to secure it with IAP.

1. Go to the IAP page in the Google Cloud Console, as shown in Figure 4-1.

   If you haven't configured your OAuth consent screen, you'll need to configure it before continuing.

2. Select the resource you want to secure by checking the box to its left.
3. On the right-side panel, click Add Member.



Identity-Aware Proxy (IAP) lets you manage who has access to services hosted on App Engine, Compute Engine, or an HTTPS Load Balancer. Learn more

To get started with IAP, add an App Engine app, a Compute Engine instance or configure an HTTPS Load Balancer .

	Resource	IAP	Method	Published	Status
☐	All Web Services				
☑	App Engine app	●	IAP	https://ruicosta-blog.uc.r.appspot.com	✓
☐	default			https://ruicosta-blog.uc.r.appspot.com	

Use external identities for authorization

Granting Owner at the *resource level* only allows modifying the access policy. It does not grant access to the application. Learn more .

Edit or delete permissions below or "Add Member" to grant new

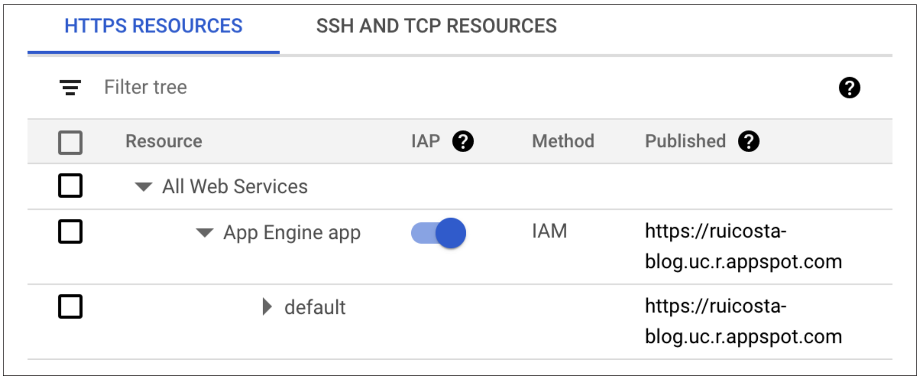
Role / Member	Inheritance
Editor (4)	
Owner (4)	

Figure 4-1. Identity-Aware Proxy configuration

4. Add the email addresses of groups or individuals to whom you want to grant access to your App Engine application. IAP Identity and Access Management (IAM) supports the following accounts:
   - Google Account
   - Google Group

- Service account
- G Suite domain

5. When you have added all the accounts that you want to provide access to your application, click Add.
6. On the IAP page, under HTTPS Resources, find the App Engine app you want to restrict access to and toggle the on/off switch in the IAP column; see Figure 4-2.



HTTPS RESOURCES | SSH AND TCP RESOURCES

Filter tree

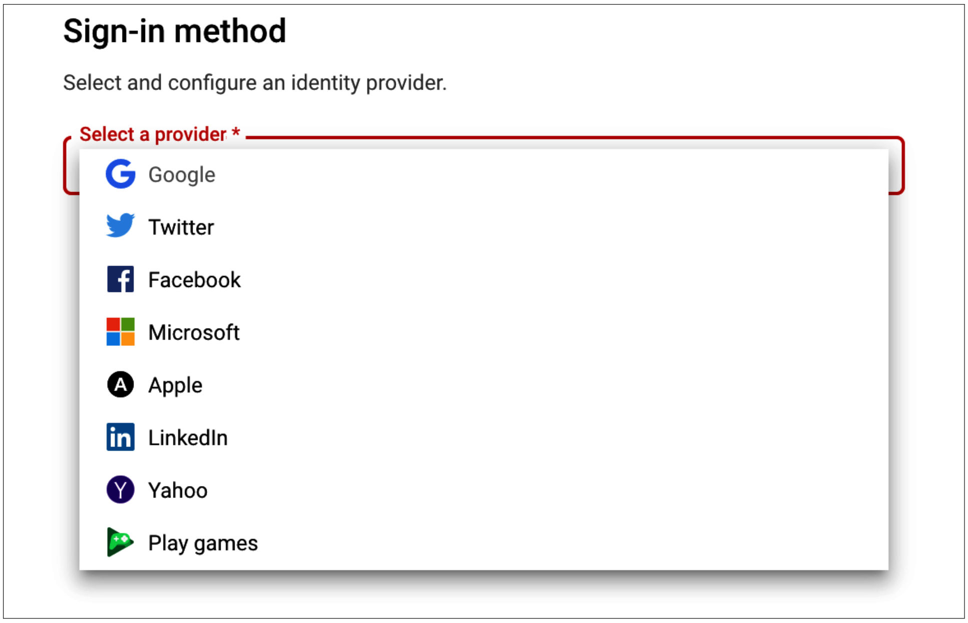
Resource	IAP	Method	Published
All Web Services			
App Engine app	●	IAM	https://ruicosta-blog.uc.r.appspot.com
default			https://ruicosta-blog.uc.r.appspot.com

Figure 4-2. Enable Identity-Aware Proxy

7. Access the URL for your App Engine application. You should be prompted to sign in. If you're not prompted, try to sign in with an Incognito window.
8. If you have authorized a user account and they sign in with the associated account, they will have full access to your application running on App Engine.
9. If you have not granted access to an account and they try to access your application, they will receive a You Don't Have Access message.

Discussion

With the Google Cloud Identity-Aware Proxy, you can restrict access to your application running on App Engine, preventing unauthorized access to your resources. The Identity-Aware Proxy also supports external identities such as Google, Microsoft, Email/Password, and others that provide a robust set of sign-in options for your users to access your application (Figure 4-3).


Sign-in method

Select and configure an identity provider.

Select a provider *

Google

Twitter

Facebook

Microsoft

Apple

LinkedIn

Yahoo

Play games

Figure 4-3. Identity-Aware Proxy providers

4.4 Mapping Custom Domains with App Engine

Problem

You want to use your own custom domain rather than the default address that App Engine provides for you.

Solution

Google Cloud provides the ability to map custom domains. It also can issue a managed certificate for SSL for HTTPS connections. You will use the cloud Hello World application created in Recipe 4.1 to enable your custom domain.

You will require a custom domain for this recipe.

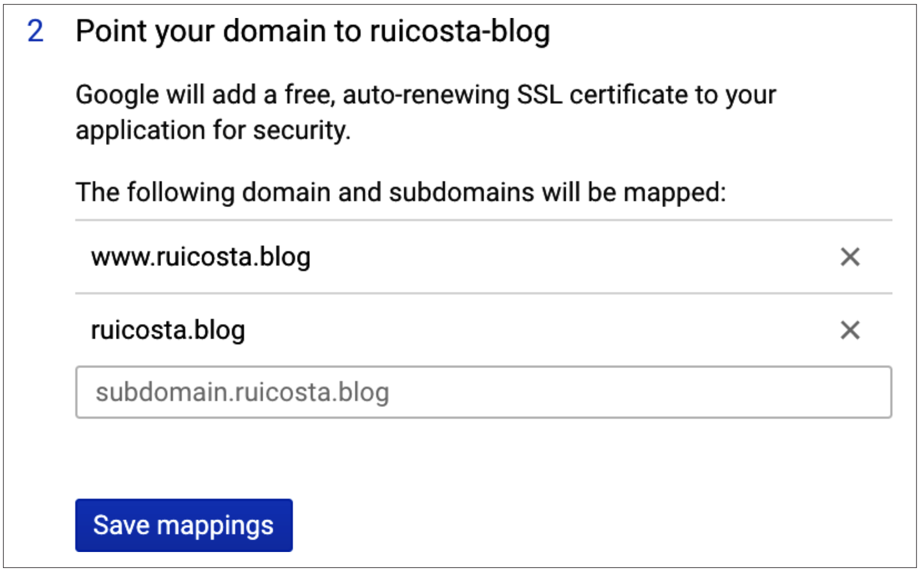
1. In the Google Cloud Console, go to App Engine > Settings > Custom Domains.
2. Click Add A Custom Domain.

3. If your domain name has been verified, the domain name will appear in the drop-down menu. Select the domain from the drop-down menu and click Continue.

   If you haven't verified your domain name, follow these steps to verify:

   a. Select Verify A New Domain from the drop-down menu.
   b. Enter your domain name and click Verify.
   c. Enter the required information on the Webmaster page.
   d. After you complete these steps on the Webmaster page, you will then return to the Add A New Custom Domain page in the Google Cloud Console.

4. In the "Point your domain to" section, add the domain or subdomain that you want to map. Click Save Mappings, as shown in Figure 4-4.



2 Point your domain to ruicosta-blog

Google will add a free, auto-renewing SSL certificate to your application for security.

The following domain and subdomains will be mapped:

www.ruicosta.blog

ruicosta.blog

Figure 4-4. Domain mapping

5. Click Continue to see your domain's DNS records.
6. Sign in to your domain registrar website and update your DNS records with the records displayed.
7. Test by opening your web browser to your newly mapped domain.

It can take several minutes for the SSL certificate to be issued.

Discussion

By mapping a custom domain, you can enable your App Engine application to align with your branding as well as keep a secure site, since Google Cloud will provide an SSL certificate for your mapped domain. Google Cloud managed certificates do not support wildcard domains; if you require wildcard domains, you will need to use self-managed certificates.

4.5 Using the Google Cloud Translation Machine Learning APIs with App Engine

Problem

You need to build a real-time translation application for voice.

Solution

Google Cloud offers a Media Translation API that adds real-time audio translation to your applications. You will build two applications, a broadcast application and a client application in this recipe, using App Engine to host your application.

Figure 4-5 demonstrates a high-level architecture of the application you will be deploying.

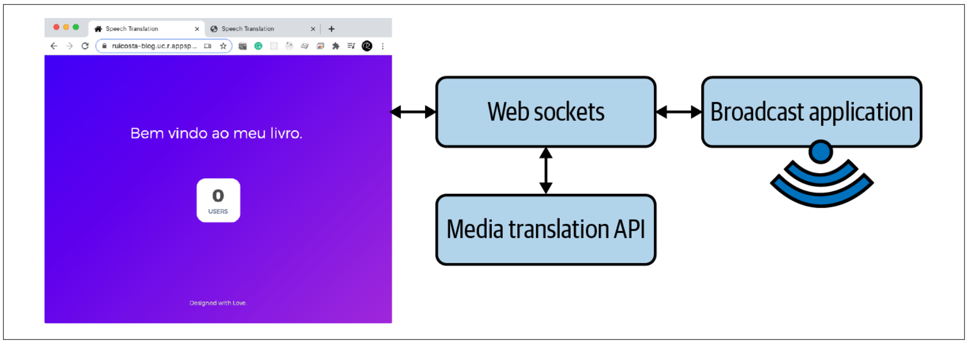


Figure 4-5. Translation application architecture

Figure 4-6 is the broadcast application the presenter uses.

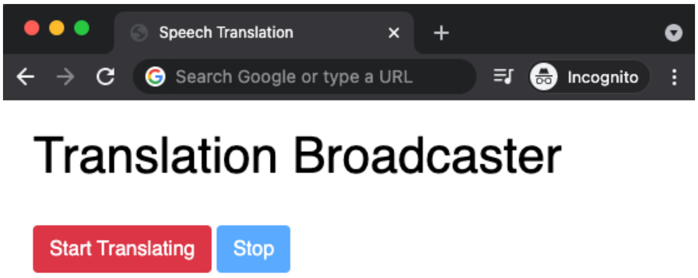


Figure 4-6. Broadcast application

Figure 4-7 is the client application where users can see the translation captions.

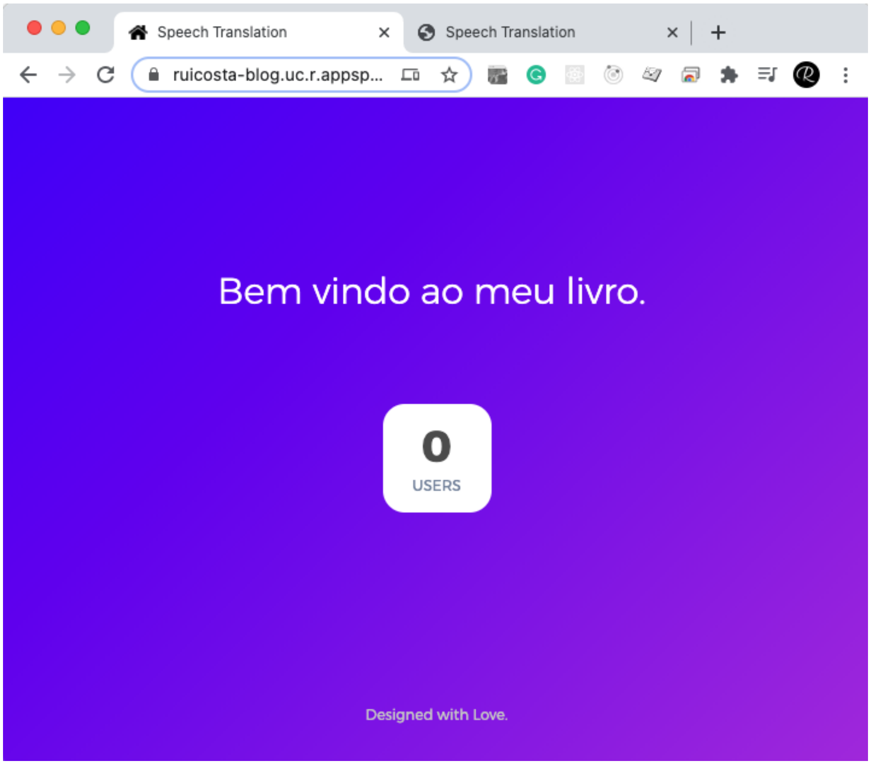


Figure 4-7. Client application

This recipe requires you to use the `git clone` command for this book's code example repository.

1. In the cloned application, go to *04-appengine/4-5-media*.
2. In your IDE, edit the *client/client.js* file and replace [PROJECT_ID] with your Google Cloud project:

   ```
   const socket = io.connect('https://[YOUR_PROJECT_ID].uc.r.appspot.com');
   ```

3. Repeat the process in step 2 but edit the broadcast/client.js file.
4. Enable the Media Translation API in the Google Cloud Console.
5. Deploy your App Engine application by running the **`gcloud app deploy`** command. In the *app.js* file in the root directory, you will notice the following Expres.js routes declared. The client path is for the users reading the translations from the person broadcasting. The person broadcasting would visit the root for the App Engine application:

   ```
   app.use('/', express.static('broadcast'))
   app.use('/client', express.static('client'))
   ```

6. Once your application is deployed, visit the root path and open a new tab with the */client* path.
7. In the broadcast application, click Start Translating and start speaking in English; watch the translation on the second tab. The translation is from English to Portuguese.
8. You can change the languages in the *app.js* file, lines 29 and 30:

   ```
     const sourceLanguage = 'en-US';
     const targetLanguage = 'pt-BR';
   ```

Discussion

In this recipe, you used an existing repository to deploy a translation application to App Engine. This application uses Express.js, WebSockets, and the Media Translation API to allow real-time audio to be translated to the language you define in the code. Since we are using WebSockets, we used App Engine Flexible, because Standard does not support WebSockets. WebSockets allowed the real-time communication of the broadcaster and users.

4.6 Building User Interfaces for Viewing Charts and Graphs

Problem

You use BigQuery as your enterprise data warehouse and need a secure method to display charts/graphs to users.

Solution

Use App Engine, along with Cube.js, BigQuery, and App Engine to build a user interface (Figure 4-8) for viewing charts and graphs from data stored in your BigQuery data set. BigQuery is a fully managed, serverless data warehouse.

You will be using Cube.js, which is an open source analytical API platform.

You will also learn how to deploy React.js to App Engine, since the user dashboards will be running with the React.js framework.

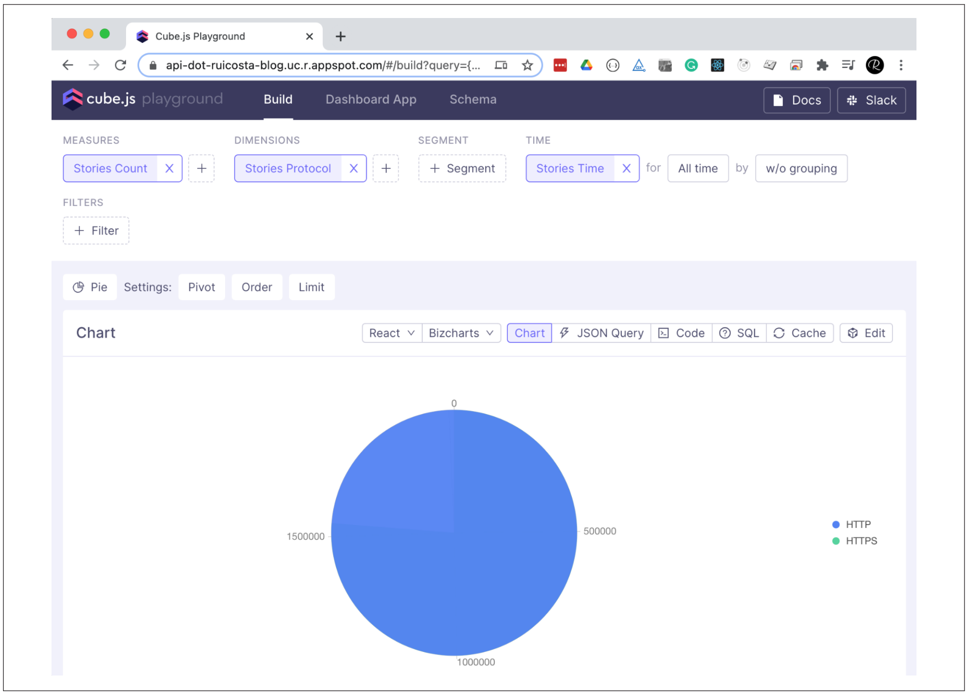


Figure 4-8. Cube.js running on App Engine

1. On your local workstation, create a temporary folder to hold the files you will need to create App Engine user dashboards.
2. In your temporary folder, using the Cube.js CLI, run the **`npx cubejs-cli create real-time-dashboard -d bigquery`** command to create a new Cube.js application for BigQuery.

3. You will need credentials to access BigQuery. In the Google Cloud Console, create a new service account. Add the BigQuery Data Viewer and BigQuery Job User roles to this service account and then generate a new JSON key file. Copy the JSON key to the root of the real-time-dashboard folder.

4. In your IDE, edit the *real-time-dashboard/.env* file to include your Google Cloud project as well as the location of your key file:

   ```
   CUBEJS_DB_BQ_PROJECT_ID=example-google-project
   CUBEJS_DB_BQ_KEY_FILE=./examples.json
   CUBEJS_DB_TYPE=bigquery
   CUBEJS_API_SECRET=SECRET
   ```

5. Cube.js uses a data schema to generate SQL code. You will be working with the BigQuery Hacker News public data set. Create a file called *Stories.js* in the *real-time-dashboard/schema* folder with the following code:

   ```
   cube(`Stories`, {
       sql: `
         SELECT *
         FROM bigquery-public-data.hacker_news.full
         WHERE type = "story" AND STARTS_WITH(UPPER(url), "HTTP")
       `,

       measures: {
         count: {
           type: `count`,
         },
       },

       dimensions: {
         protocol: {
           sql: `UPPER(REGEXP_EXTRACT(${CUBE}.url, r"^([a-zA-Z]+):"))`,
           type: `string`,
         },

         time: {
           sql: `timestamp`,
           type: `time`,
         },
       },
     });
   ```

6. Now run a real-time dashboard locally to test and validate that it's working as expected. Run the **`npm run dev`** command in the *real-time-dashboard* folder.

7. In your browser, go to *http://localhost:4000*, which should launch Cube.js Playground, as shown in Figure 4-9.

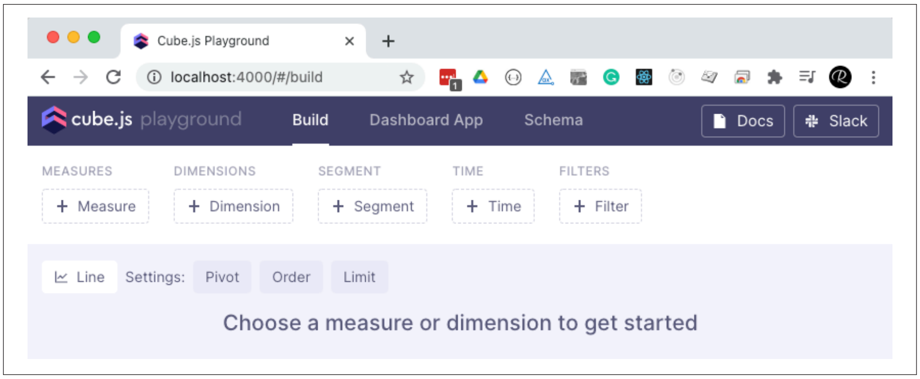


Figure 4-9. Cube.js running locally

8. To test that it's connecting to BigQuery, click Measure and choose Stories Count, as shown in Figure 4-10.

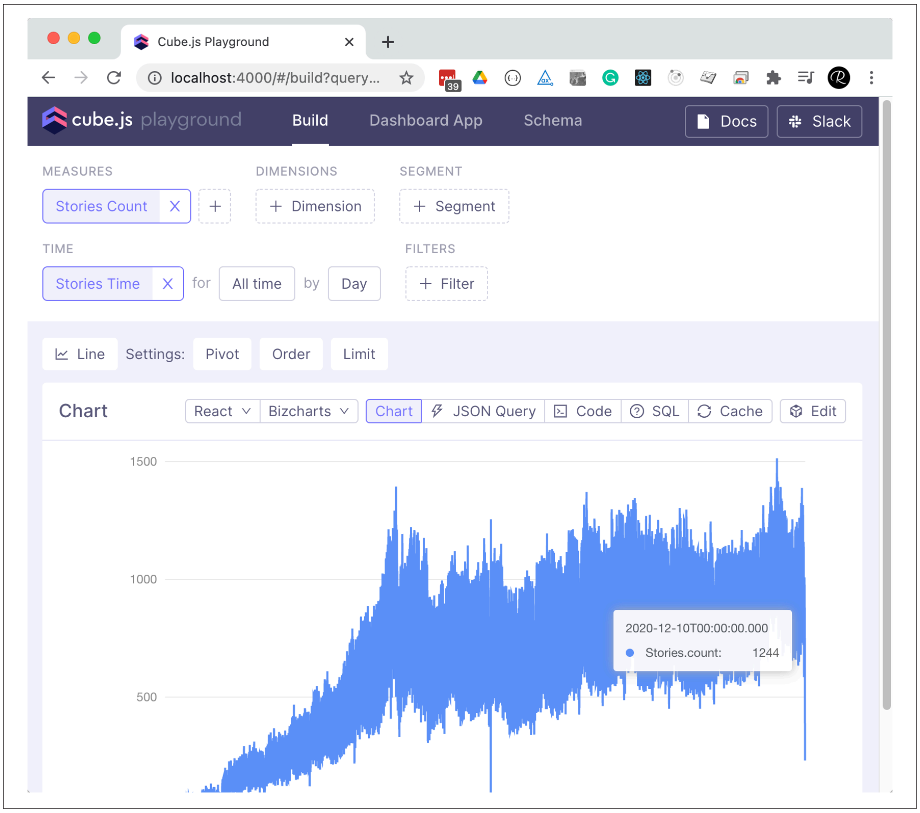


Figure 4-10. Cube.js connection to BigQuery

9. This service will become an API running on App Engine. The user dashboard will connect to the Cube.js API to fetch the required data and visualize it for the user.
10. To build the dashboard, click the Dashboard App in the Cube.js Playground.
11. Once it's installed, locate the folder in your IDE; it will be under *real-time-dashboard/dashboard-app*.
12. In your IDE, edit the *src/pages/DashboardPage.js* to replace the following line:

    ```
    const DashboardItems = []
    ```

    with:

    ```
    const DashboardItems = [
      {
        id: 0,
        name: "Orders Status by Customers City",
        vizState: {
          query: {
            "measures": [
              "Stories.count"
            ],
            "timeDimensions": [],
            "order": {
              "Stories.count": "desc"
            },
            "dimensions": [
              "Stories.protocol"
            ]
          },
          chartType: "pie",
        }
      },
      {
        id: 1,
        name: "Orders Status by Customers City",
        vizState: {
          query:   {
            "measures": [
              "Stories.count"
            ],
            "timeDimensions": [
              {
                "dimension": "Stories.time",
                "granularity": "year"
              }
            ],
            "order": {},
            "dimensions": []
          },
          chartType: "line",
    ```

```
    }
  },
];
```

13. In your IDE, edit *src/components/ChartRenderer.js* to include the following:

```
 const ChartRenderer = ({
-   vizState
+   vizState, cubejsApi
-   const renderProps = useCubeQuery(query);
+   const renderProps = useCubeQuery(query, { subscribe: true, cubej-
sApi });
```

> Remove the lines in *CharRenderer.js* that are noted with a subtraction symbol (-) and add the lines noted with the addition symbol (+).
>
> To learn more about Cube.js and React, please visit the online reference.

14. Create an *app.yaml* file in *real-time-dashboard/dashboard-app* with the following code:

```
runtime: nodejs14
handlers:
- url: /(.*\..+)$
  static_files: build/\1
  upload: build/(.*\..+)$
- url: /.*
  static_files: build/index.html
  upload: build/index.html
```

15. This configuration lets App Engine serve the optimized React.js build. When making changes, you will always need to run an npm run build before deploying your new version to App Engine.
16. Run the following commands to deploy the Dashboard to App Engine:

```
npm run build
    gcloud app deploy
```

17. After this has been successfully deployed, run the **`gcloud app browse`** command to view the application in your browser.
18. Copy the URL of your deployed dashboard app and edit the *real-time-dashboard/dashboard-app/App.js* file to replace the `const API_URL` variable with yours. It should look like this:

```
const API_URL = "https://ruicosta-blog.uc.r.appspot.com";
```

19. Go ahead and redeploy:

```
npm run build
    gcloud app deploy
```

20. At this point, the Dashboard is ready to connect to the Cube.js API that you just updated in the *App.js* file. Now it's time to deploy the API to App Engine.
21. Create a Dockerfile in the *real-time-dashboard* folder with the following code:

    ```
    FROM cubejs/cube:latest

    COPY . .
    ```
22. Create an *app.yaml* file in the *real-time-dashboard* folder with the following code:

    ```
    runtime: custom
    env: flex
    service: api
    ```
23. Since Cube.js uses WebSockets and App Engine Standard does not support WebSockets, we need to use a custom runtime, so you will use Flexible for the API.
24. Update the content of the *cube.js* file with the following, located in the root of the *real-time-dashboard* folder:

    ```
    module.exports = {
        processSubscriptionsInterval: 1,
        orchestratorOptions: {
          queryCacheOptions: {
            refreshKeyRenewalThreshold: 1,
          }
        },
      };
    ```
25. Update the content of the *.env* file with the following, located in the root of the *real-time-dashboard* folder, to include `CUBEJS_WEB_SOCKETS=true`.
26. Create a new file, called *dispatch.yaml*, in the root of the *real-time-dashboard* folder:

    ```
      - url: "*/cubejs-api*"
        service: api
    ```

    The *dispatch.yaml* file allows you to override routing rules and allows your Dashboard application to access the API via the main URL of the Dashboard so as not to cause issues with CORS.
27. You are now ready to deploy the API and have users access data via the Dashboard. In the root of the real-time-dashboard, run the **`gcloud app deploy`** command to deploy the API.
28. Once this has completed, deploy the dispatch rules by running **`gcloud app deploy dispatch.yaml`**.
29. If you now access the URL of the Dashboard, you should be able to see what's shown in Figure 4-11.

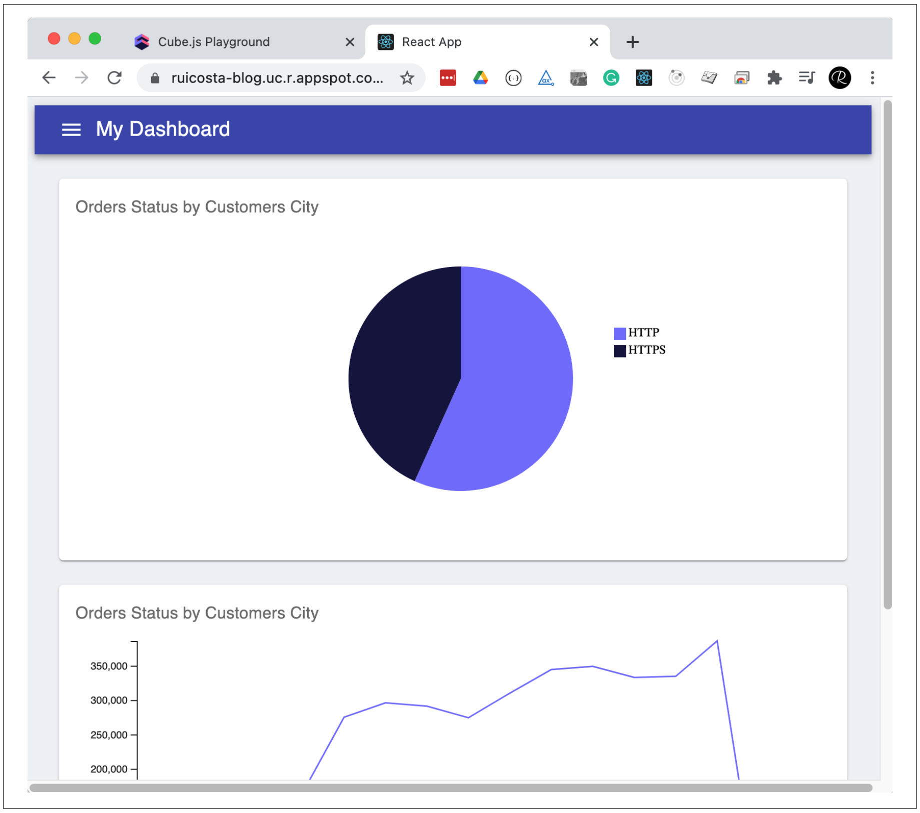

Cube.js Playground
React App
ruicosta-blog.uc.r.appspot.co...
My Dashboard
Orders Status by Customers City
HTTP
HTTPS
Orders Status by Customers City
350,000
300,000
250,000
200,000

Figure 4-11. Cube.js running on App Engine

30. Don't forget to secure your application by enabling IAP.

Discussion

In this recipe, you deployed the Cube.js API, a user Dashboard running on the React.js Framework, to App Engine and created routes with dispatch rules to build an interactive real-time dashboard for users.

There are many moving parts in this recipe, but the key takeaways are:

- App Engine Standard does not support WebSockets, so you used App Engine Flexible because the Cube.js API relies on WebSockets.
- App Engine is very flexible; with custom runtimes, the possibilities of running your application on App Engine are endless.

4.7 Debugging an Instance

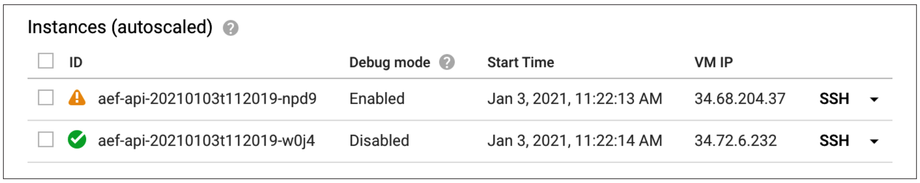
Problem

You notice an issue with your application, and you need a way to access the logs to debug.

Solution

With App Engine Flexible, enable the debug mode. While debugging is enabled, you can access the VM to view the log files of your custom runtime.

1. To enable debug mode, run the gcloud `app --project PROJECT_ID` command.
2. It will prompt you with the instances available to enable debugging. Choose one.
3. In the Google Cloud Console, choose App Engine > Instances.
4. You should notice in the instance you chose that Debug mode (Figure 4-12) is now enabled.



Instances (autoscaled)

	ID	Debug mode	Start Time	VM IP	
☐	⚠ aef-api-20210103t112019-npd9	Enabled	Jan 3, 2021, 11:22:13 AM	34.68.204.37	SSH ▾
☐	✓ aef-api-20210103t112019-w0j4	Disabled	Jan 3, 2021, 11:22:14 AM	34.72.6.232	SSH ▾

Figure 4-12. App Engine Debug mode enabled

5. Click the SSH button to connect to the instance.
6. At this point, you are connected to the instance host, which has several containers running in it.

In addition to your container running on App Engine Flexible, you will also have three additional containers:

- Fluentd Logging agent
- Memcache proxy agent
- NGINX proxy

7. Run **`sudo docker ps`** to list the containers running.
8. The output of the `sudo docker ps` command lists each container; locate the row that contains your project ID and note the NAME of this container.
9. To view the logs, run the **`sudo docker logs [CONTAINER-NAME]`** command.
10. This allows you to view the logs from your application for debugging purposes.
11. You can also connect to the instance by running `sudo docker exec -it CONTAINER_NAME /bin/bash`.
12. When completed, don't forget to disable debugging by running the `gcloud app --project PROJECT_ID instances disable-debug` command.

Discussion

The ability to connect to an instance and its containers allows you to debug your application running on App Engine Flexible.

4.8 Using CI/CD

Problem

You need a method to automate the deployment of your application to App Engine every time a change is made to the source code.

Solution

Use GitLab CI/CD, a tool that allows you to apply continuous integration (CI), continuous delivery (CD), and continuous deployment (also CD) to your application.

1. Create a new GitLab project and clone the new repository to your local machine.
2. Create the Hello World application from Recipe 4.1, but do not deploy it to App Engine.
3. In the root of the directory, create a GitLab CI/CD file named *.gitlab-ci.yml* with the following contents:

```
    image: google/cloud-sdk:slim
deploy:
    stage: deploy
    environment: Production
    only:
        -  master
    script:
        - gcloud auth activate-service-account --key-file $GOOGLE_SER-
VICE_ACCOUNT_FILE
        - gcloud app deploy app.yaml --quiet --project $GOO-
GLE_PROJECT_ID --version 1
```

4. In the Google Cloud Console, go to Identity > Service Accounts.
5. Click Create Service Account.
6. Enter name and description and then click Create.
7. Select the Editor role and click Continue.
8. Select the service account you just created and, in Options, click Create A Key In JSON Format. Download the key to your local workstation.
9. In the GitLab console within your project, go to Settings > CI/CD.
10. Expand the Variables section.
11. Create a new variable.
12. Change the type of variable to File. The key will be named *GOOGLE_SERVICE_ACCOUNT_FILE*, and the value will be the content of the file that has been previously downloaded.
13. Create another variable, named *GOOGLE_PROJECT_ID*, and the value will be the ID of the Google Cloud project.
14. Commit your code and deploy your application to App Engine. In addition, commit your changes to your GitLab repository.
15. In the GitLab console, open the GitLab CI/CD page. You will notice your pipeline running.
16. If you click the pipeline, you will see the deployment steps; note Job Succeeded.

Discussion

The continuous methodologies of software development are based on automating the tests and deployments of your source code to minimize the chance of errors. GitLab CI/CD provides a set of tools, including the continuous deployment methodology used in this recipe. Now, instead of deploying your application manually, it can be deployed automatically to Google Cloud App Engine.

CHAPTER 5

Google Cloud Compute Engine

Google Cloud Compute Engine provides you the ability to run virtual machines on Google's infrastructure. You can run Windows and Linux virtual machines. You can customize the virtual machine to meet your needs. You can change the virtual machine's memory allocation, you can change how many virtual CPUs are assigned, and you can even automate patching of the operating system.

This chapter contains recipes for creating and managing your virtual machines. You'll find recipes on unique methods to automate deployments, deploy containers to virtual machines, and use Identity-Aware Proxy (IAP) to tunnel Remote Desktop Protocol (RDP) traffic to connect to your Windows virtual machines securely.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.

5.1 Creating a Windows Virtual Machine

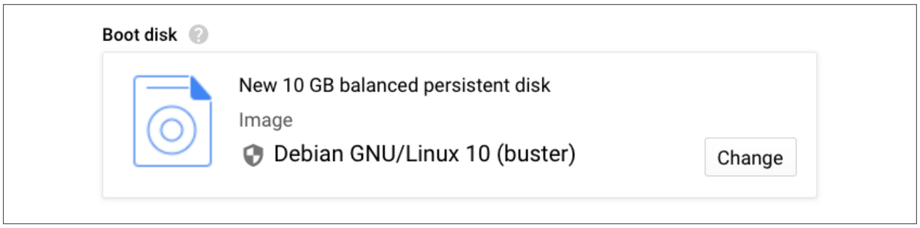
Problem

You have an application that needs to be installed on a Windows server. You also need access to the operating system to change configuration options required by the application.

Solution

Using the Google Cloud Console, create a Windows server on Google Cloud Compute Engine. This will provide you with full access to the operating system to allow you to make any configuration changes required by the application.

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Compute Engine.
3. Select VM Instances from the menu and click Create.
4. Choose a name for your instance.
5. Choose a region and zone for where this VM will be hosted.
6. Select a machine configuration or customize based on your requirements.
7. Click the Change button in the Boot Disk section, as shown in Figure 5-1.


Boot disk

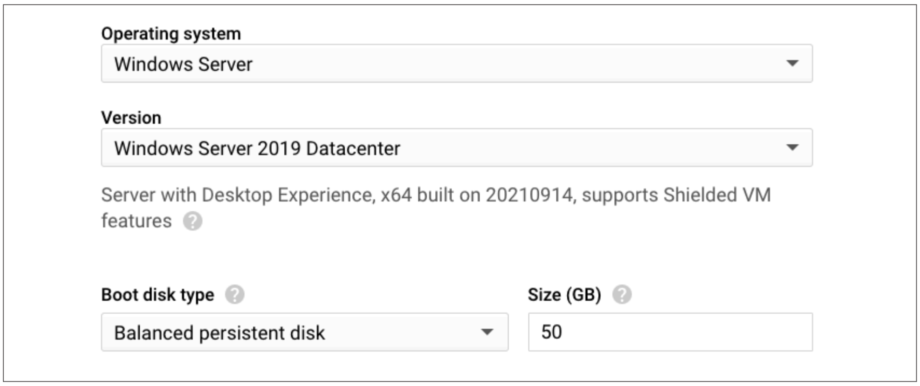
New 10 GB balanced persistent disk

Image

Debian GNU/Linux 10 (buster)

Figure 5-1. Boot disk

8. Choose Windows Server for the operating system.
9. Choose the Windows version required and select a disk size, as shown in Figure 5-2.


Operating system: Windows Server

Version: Windows Server 2019 Datacenter

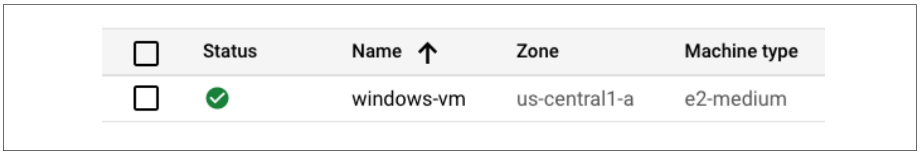
Server with Desktop Experience, x64 built on 20210914, supports Shielded VM features

Boot disk type: Balanced persistent disk

Size (GB): 50

Figure 5-2. Windows version selection

10. Click Select.
11. Leave other settings to default and click Create.
12. It will take a few seconds for your Windows virtual machine to start up. Figure 5-3 shows the instance successfully started.



☐	Status	Name ↑	Zone	Machine type
☐	✓	windows-vm	us-central1-a	e2-medium

Figure 5-3. Windows instance successfully started

Discussion

In this recipe, you used the Google Cloud Console to create a Windows server. Creating a Windows server on Compute Engine is a quick and painless way to run your Windows workloads on a managed infrastructure. Explore the many options available to customize the memory and virtual CPU allocation for your virtual machine. You are never locked into a preconfigured template.

5.2 Creating a Linux Virtual Machine and Installing NGINX

Problem

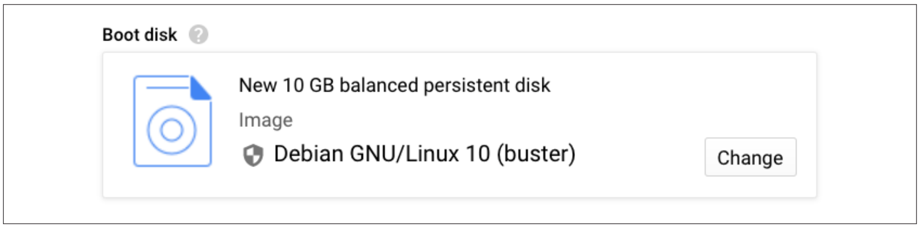
You want to install an NGINX web server in the cloud and need full access to the operating system hosting NGINX.

Solution

Using the Google Cloud Console, create a Linux virtual machine on Google Cloud Compute Engine. This will provide you with full access to the operating system to allow you to make any configuration changes the application requires. You will also connect to the instance and install an NGINX web server.

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Compute Engine.
3. Select VM Instances from the menu and click Create.
4. Choose a name for your instance.
5. Choose a region and zone for where this VM will be hosted.
6. Select a machine configuration or customize based on your requirements.

7. Leave Boot Disk set to Debian GNU/Linux 10 (buster), as shown in Figure 5-4.


Boot disk

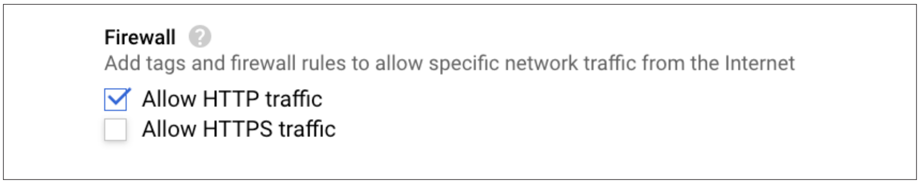
New 10 GB balanced persistent disk

Image

Debian GNU/Linux 10 (buster)

Figure 5-4. Boot disk

8. Select Allow HTTP Traffic to the instance, as shown in Figure 5-5.

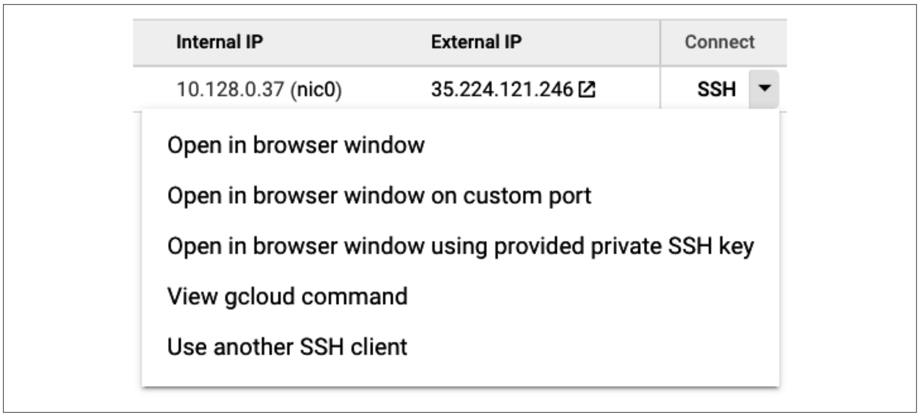

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

- [x] Allow HTTP traffic
- [ ] Allow HTTPS traffic

Figure 5-5. Allow HTTPS traffic

9. Click Create.
10. Once the instance has been created and the public IP address has been set, click SSH and select Open In The Browser Window, as shown in Figure 5-6.



Internal IP	External IP	Connect
10.128.0.37 (nic0)	35.224.121.246	SSH

Open in browser window
Open in browser window on custom port
Open in browser window using provided private SSH key
View gcloud command
Use another SSH client

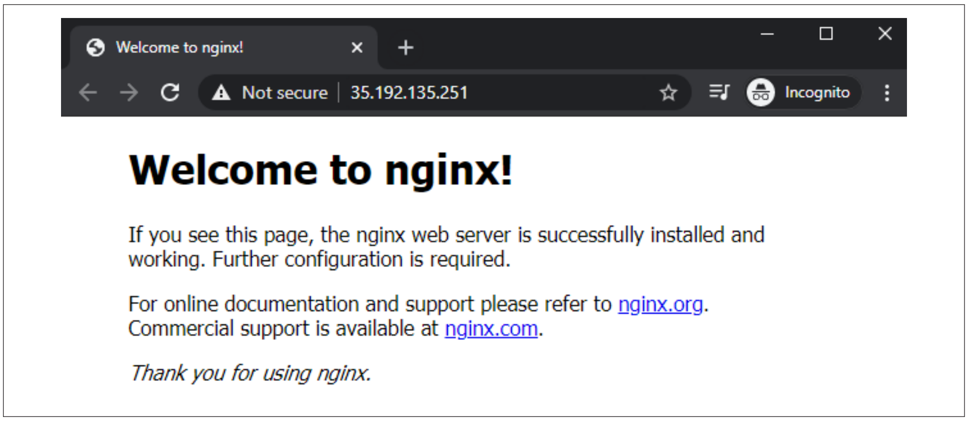
Figure 5-6. SSH menu

11. In the instance terminal, enter the following commands to install NGINX:

```
sudo su -
apt-get update
apt-get install -y nginx
service nginx start
```

12. In the Cloud Console, click the public IP address of your server to open the URL in a new browser tab.

In your browser, you will see a screen that looks like Figure 5-7.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 5-7. NGINX running on Google Compute Engine

Discussion

In this recipe, you used the Google Cloud Console to create a Linux virtual machine. You also installed NGINX via the Google Cloud SSH browser. The SSH browser is a quick and easy way to get access to your Linux terminal without having to manage local SSH keys, which you can do if you choose. You also allowed HTTP traffic to the instance by selecting Allow HTTP Traffic in the firewall section of the creation screen. For this to work, you need to make sure you do not delete the default firewall rule that allows HTTP traffic. If you happen to delete the firewall rule, you can re-create it or create a new one and associate the firewall tags with the instances you need HTTP traffic to be allowed for.

5.3 Connecting to Your Windows Virtual Machines with Identity-Aware Proxy TCP Forwarding

Problem

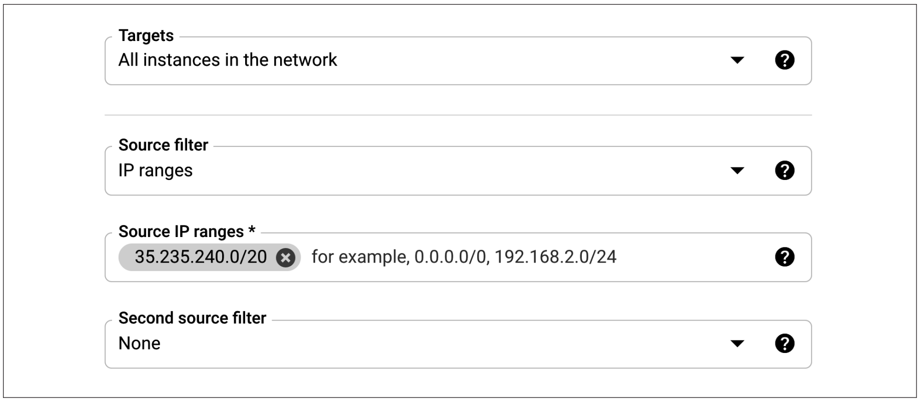
You want to securely connect to your Windows Virtual Instance running in Google Cloud.

Solution

Using Google Cloud Identity-Aware Proxy (IAP) with RDP, you will be able to securely connect to your Virtual Instance running Microsoft Windows. Using IAP allows you to also connect to the instance even if it did not have a Public IP address; this is accomplished through the use of TCP forwarding. IAP also lets you control who can access the instance.

1. You will need an existing Windows virtual machine ready and running on Compute Engine. You can reference Receipe 5.1 to install a Microsoft Windows Server in Google Cloud.
2. In the main menu of the Google Cloud Console, navigate to Networking and click VPC Network.
3. Click Firewall in the menu.
4. Click Create A Firewall Rule.
5. Enter a name for the firewall rule as **`allow-remote-iap`**.
6. For the Targets, select All Instances In The Network.
7. For Source IP Ranges, enter the following CIDR range, as shown in Figure 5-8: **`35.235.240.0/20`**.

The 35.235.240.0/20 CIDR range contains all IP addresses that Google Cloud IAP uses for TCP forwarding.


Targets
All instances in the network

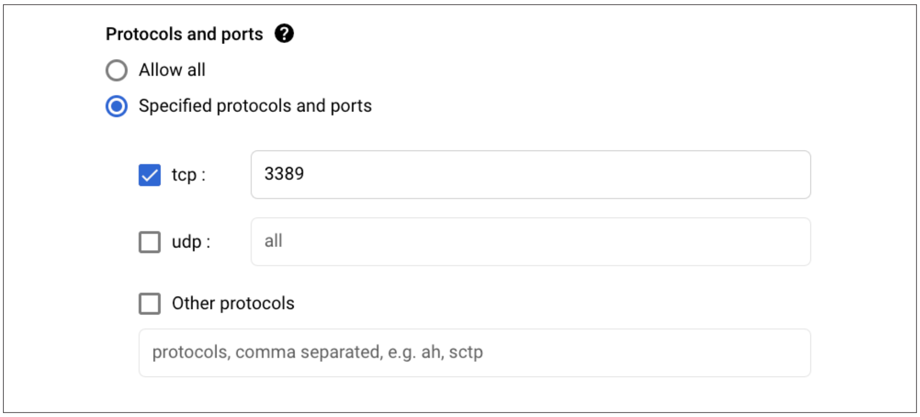
Source filter
IP ranges

Source IP ranges *
35.235.240.0/20 for example, 0.0.0.0/0, 192.168.2.0/24

Second source filter
None

Figure 5-8. Source IP ranges

8. Select TCP and enter 3389 to allow RDP in the Protocols And Ports section, as shown in Figure 5-9.


Protocols and ports
Allow all
[✓ Specified protocols and ports]

[✓ tcp :] 3389

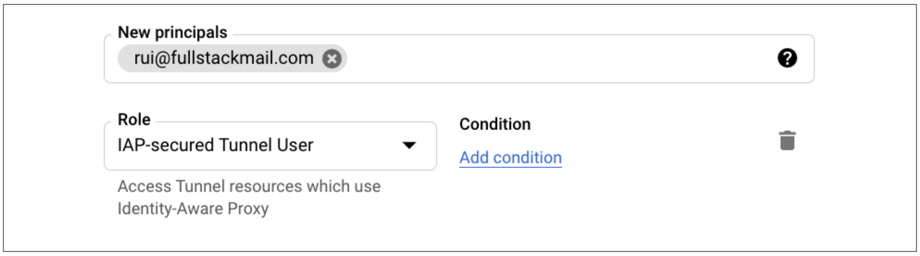
udp : all

Other protocols

protocols, comma separated, e.g. ah, sctp

Figure 5-9. Protocols and ports

9. Click Create.
10. Navigate to IAM & Admin > IAM in the Google Cloud Console menu.
11. Click Add to allow the groups and users to access IAP TCP forwarding, which will grant them access to connect to the instances in this project.
12. For the members, select the groups and or users to whom you want to grant access.
13. Select a role and choose IAP-Secured Tunnel User, as shown in Figure 5-10.



New principals

rui@fullstackmail.com

Role

IAP-secured Tunnel User

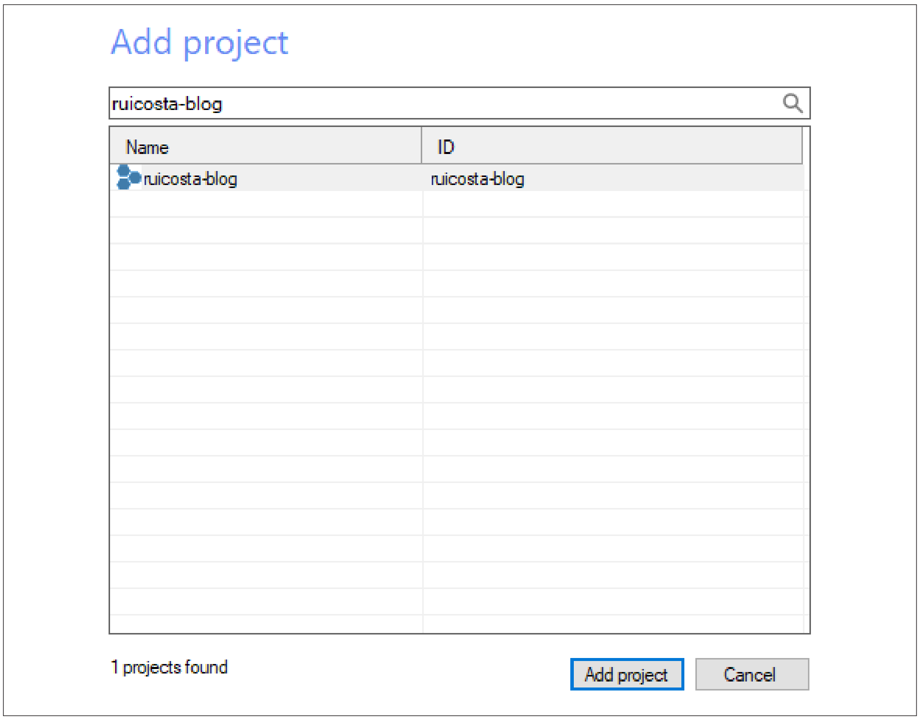
Access Tunnel resources which use Identity-Aware Proxy

Condition

Add condition

Figure 5-10. IAP secured tunnel user

14. Click Save.
15. Install the IAP Desktop application; you can find the download at *https://github.com/GoogleCloudPlatform/iap-desktop*. The IAP Desktop application allows you to manage multiple RDP and SSH connections to Google Compute instances securely with IAP.
16. Launch the IAP Desktop application and sign-in with the account you granted access to in step 13.
17. Select your Google Cloud project, as shown in Figure 5-11.



Add project

ruicosta-blog

Name	ID
ruicosta-blog	ruicosta-blog

1 projects found

Figure 5-11. IAP Desktop, Add project

18. Once successfully authenticated, you should see your Windows virtual machines listed, as shown in Figure 5-12.

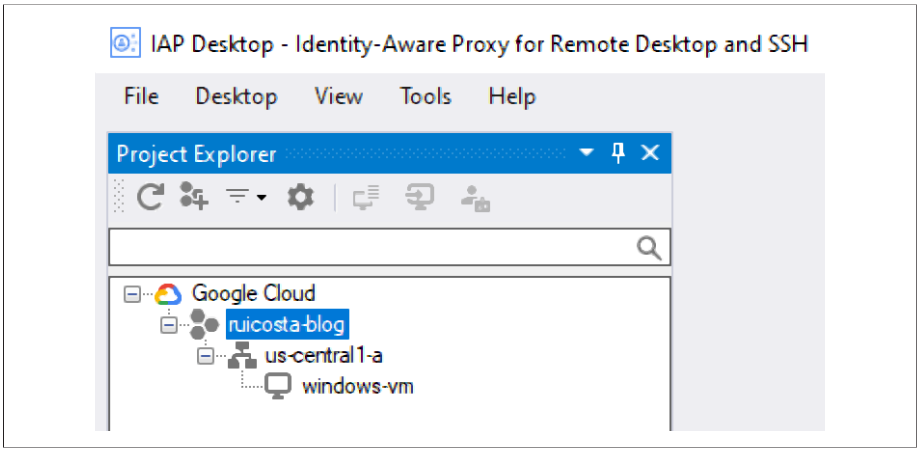


Figure 5-12. IAP Desktop, Add project

19. Right-click the instance and select Connect; after successfully authenticating, you should see the Windows Server desktop, as shown in Figure 5-13.

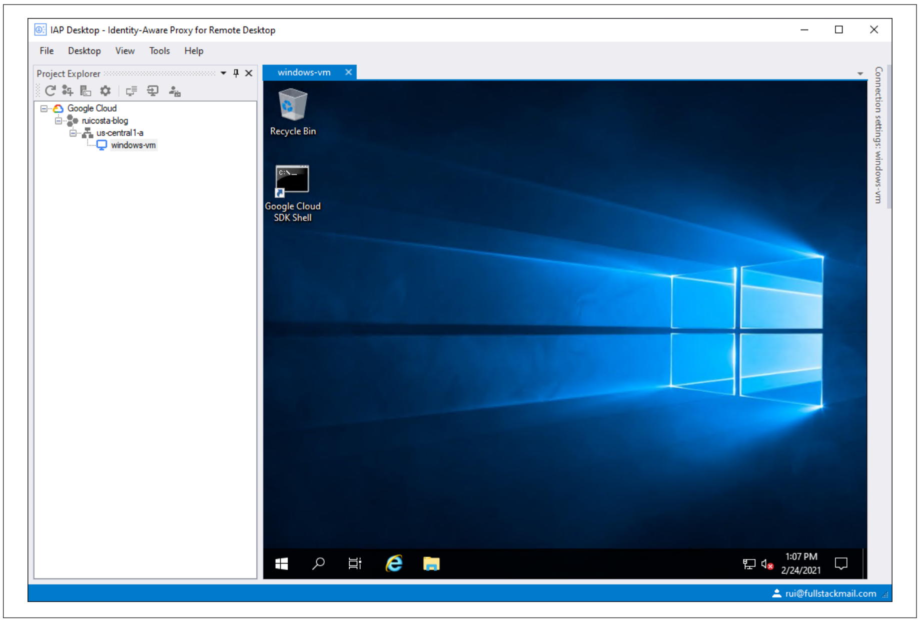


Figure 5-13. IAP Desktop, full view

Discussion

IAP allows you to create a centralized authorization layer for applications. In this recipe, you used IAP to tunnel RDP traffic to your Windows virtual machines. Using the IAP Desktop client, you established a secure connection to Google Cloud, and then your account was authenticated and authorized. Once successfully authenticated and authorized, you then had access to the instances you had been granted in your Google Cloud project. You can further secure this process by implementing two-step verification; see Recipe 5.4 for more information. Using the IAP Desktop application also provides the following benefits:

- You do not need to expose SSH or RDP to the public internet.
- You can connect to virtual machines that do not have a public IP address.

5.4 Securing Your Virtual Machine Logins with Two-Step Verification

Problem

You want to secure your virtual machine OS logins with a two-step verification process.

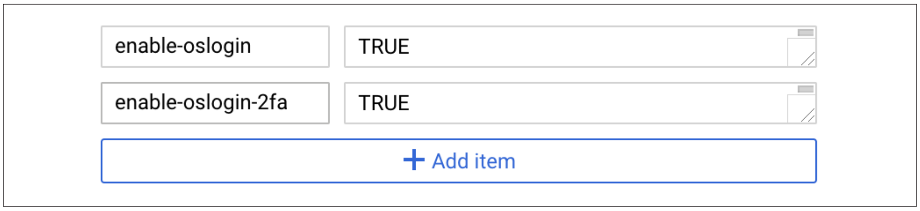
Solution

Compute Engine provides the ability to secure your virtual machine logins with a two-step verification process. In this recipe, you will enable two-step verification on your Google Account as well as on a virtual machine to force users to provide a second method of authentication besides their password as a text message, phone prompt, or Google Authenticator.

OS Login has some limitations; please review the latest documentation for updates to these limitations.

1. You will need to have a Linux virtual machine running on Compute Engine to continue with this recipe.
2. You will also need to enable two-step verification on your Google Account from your web browser.
3. Sign in to Google Cloud Console.

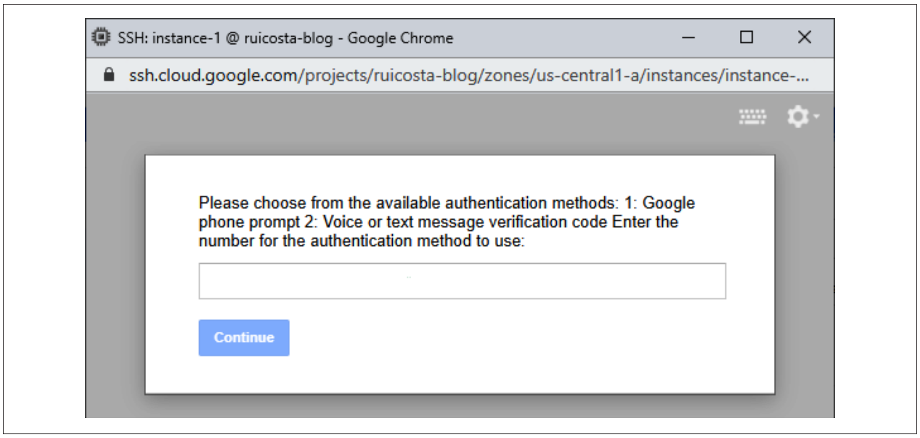
4. In the main menu, navigate to Compute and click Compute Engine.
5. Select VM Instances from the menu and select the instance you want to enable the two-step verification for.
6. Click Edit.
7. In the Custom Metadata section, add the following key/value pairs, as shown in Figure 5-14:
   - enable-oslogin: TRUE
   - enable-oslogin-2fa: TRUE



enable-oslogin	TRUE
enable-oslogin-2fa	TRUE

Figure 5-14. Custom metadata

8. Click Save.
9. Add the Compute OS Admin Login or Compute OS Login role to the user account you wish to grant access to for the virtual machine.
10. Using the Cloud SSH browser, connect to the instance.
11. You will now be prompted for a two-step verification, as shown in Figure 5-15.



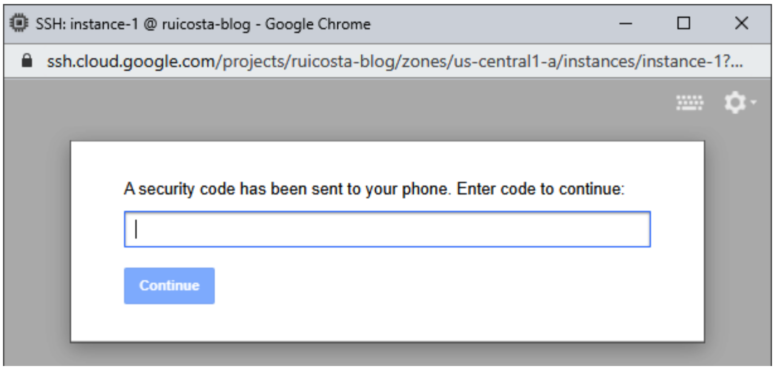
SSH: instance-1 @ ruicosta-blog - Google Chrome

ssh.cloud.google.com/projects/ruicosta-blog/zones/us-central1-a/instances/instance-...

Please choose from the available authentication methods: 1: Google phone prompt 2: Voice or text message verification code Enter the number for the authentication method to use:

Figure 5-15. Two-step verification

12. After making your selection, you should receive a code that you will enter in the two-step verification, as shown in Figure 5-16.


SSH: instance-1 @ ruicosta-blog - Google Chrome

ssh.cloud.google.com/projects/ruicosta-blog/zones/us-central1-a/instances/instance-1?...

A security code has been sent to your phone. Enter code to continue:

Figure 5-16. Security code

13. Once verification is complete, you will have access to Linux virtual machines.

Discussion

By implementing a two-step verification, you further secured access to your virtual machines running in Google Cloud. Users who need access to instances running in Google Cloud with two-step verification enabled will be required to enter a code or other methods as physical keys to access the instances. The two-step verification process is set on your Google account.

5.5 Running Startup Scripts

Problem

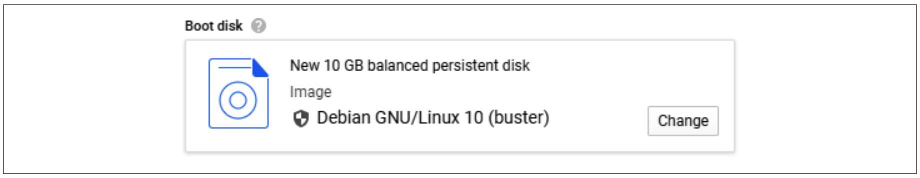
You want to install an NGINX web server in the cloud, and you need the ability to replicate this process multiple times to deploy additional instances.

Solution

Using the Google Cloud Console, create a Linux virtual machine on Google Cloud Compute Engine. You will also create a startup script to automate the installation of NGINX so you can quickly deploy additional instances.

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Compute Engine.

3. Select VM Instances from the menu and click Create.
4. Choose a name for your instance.
5. Choose a region and zone for where this VM will be hosted.
6. Select a machine configuration or customize based on your requirements.
7. Leave the Boot Disk set to Debian GNU/Linux 10 (buster), as shown in Figure 5-17.

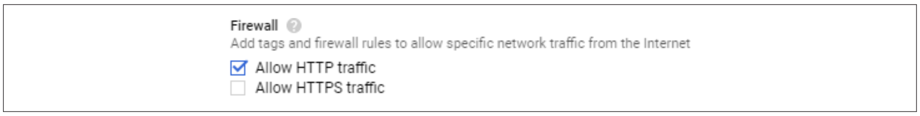


Boot disk

New 10 GB balanced persistent disk
Image
Debian GNU/Linux 10 (buster)

Figure 5-17. Boot disk

8. Select Allow HTTP Traffic to the instance, as shown in Figure 5-18.

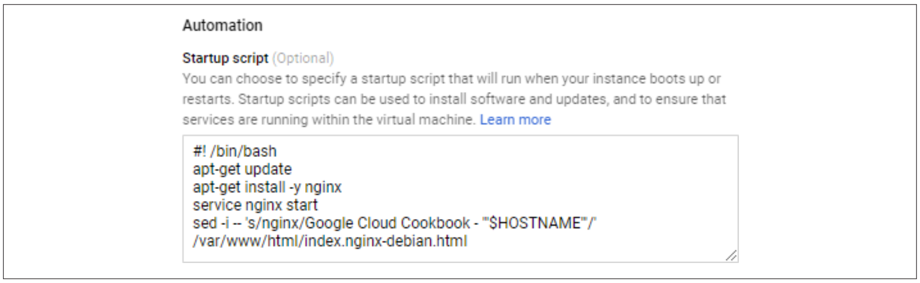


Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
☑ Allow HTTP traffic
☐ Allow HTTPS traffic

Figure 5-18. Allow HTTP Traffic

9. In the Startup script text input, enter the following commands; Figure 5-19 shows an example:

```
#! /bin/bash
apt-get update
apt-get install -y nginx
service nginx start
sed -i -- 's/nginx/Google Cloud Cookbook - '"$HOSTNAME"'/' /var/www/
html/index.nginx-debian.html
```



Automation

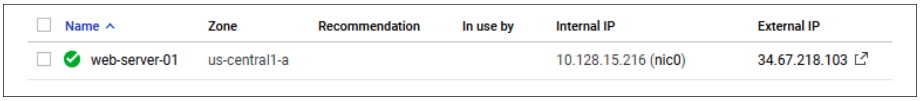
Startup script (Optional)
You can choose to specify a startup script that will run when your instance boots up or restarts. Startup scripts can be used to install software and updates, and to ensure that services are running within the virtual machine. Learn more

```
#! /bin/bash
apt-get update
apt-get install -y nginx
service nginx start
sed -i -- 's/nginx/Google Cloud Cookbook - '"$HOSTNAME"'/'
/var/www/html/index.nginx-debian.html
```

Figure 5-19. Automation script

10. Click Create.

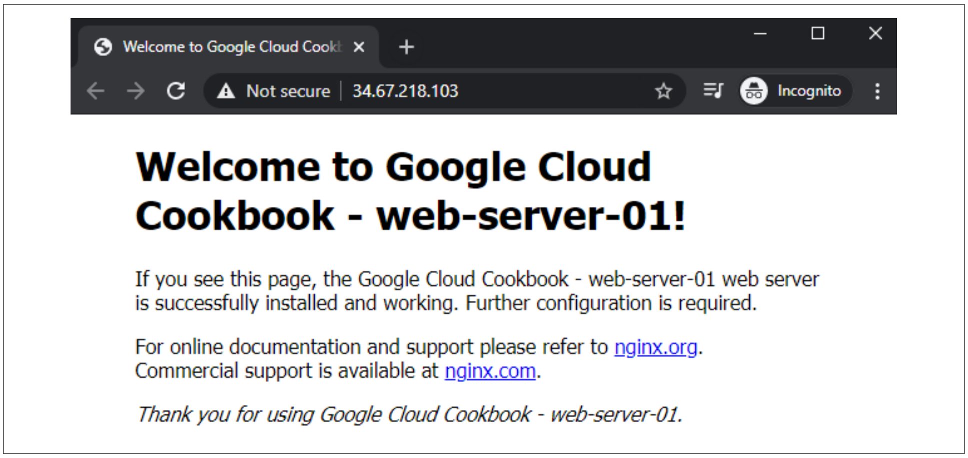
11. Once the instance has been created and the public IP address has been set, open your internet browser to the instance's public IP address, as shown in Figure 5-20.



Name	Zone	Recommendation	In use by	Internal IP	External IP
web-server-01	us-central1-a			10.128.15.216 (nic0)	34.67.218.103

Figure 5-20. External public IP address

In your browser, you will see a screen that looks like Figure 5-21.



Welcome to Google Cloud Cookbook - web-server-01!

If you see this page, the Google Cloud Cookbook - web-server-01 web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using Google Cloud Cookbook - web-server-01.

Figure 5-21. NGINX running on Google Compute Engine

Discussion

Compute Engine allows you to create startup and shutdown scripts for your virtual machines. With this recipe, you had the opportunity to use startup scripts to automate the installation of software. Using startup scripts automated the deployment of a large fleet of web servers. In Recipe 5.6, you will learn how to use the startup scripts to deploy a cluster of web servers, which makes the process of installing software more efficient than connecting to each instance to install the software. For additional information on the options for startup scripts, please review the documentation.

5.6 Creating a Group of NGINX Web Servers with a Managed Instance Group

Problem

You need to host a highly available web server, using NGINX. If one server fails, you need your web application to tolerate a host failure.

Solution

Using managed instance groups (MIGs), you will create a grouping of Linux virtual machines on Google Cloud Compute Engine. You will also create network load balancers to distribute traffic across the virtual machines.

1. Sign in to Google Cloud Console and launch Cloud Shell.
2. In Cloud Shell, create a startup script named `nginx-startup.sh` with the following commands:

   ```
   #! /bin/bash
   apt-get update
   apt-get install -y nginx
   service nginx start
   sed -i -- 's/nginx/Google Cloud Cookbook - '"$HOSTNAME"'/' /var/www/
   html/index.nginx-debian.html
   ```

3. Create a new instance template with the gCloud command and define the startup script parameter to the file created in step 2:

   ```
   gcloud compute instance-templates create nginx-template \
   --metadata-from-file startup-script=nginx-startup.txt
   ```

   An instance template defines the machine type as the Boot Disk image, labels, and other instance properties. You can use this template to create a MIG or even a virtual machine instance.

4. Create a target pool so you can have a single access point for load balancing. Run the following command:

   ```
   gcloud compute target-pools create nginx-pool
   ```

5. Run the following command to create an instance group:

   ```
   gcloud compute instance-groups managed create nginx-group \
            --base-instance-name nginx \
            --size 2 \
            --template nginx-template \
           --target-pool nginx-pool
   ```

6. Open the Cloud Console and navigate to VM Instances; you should see two new instances created from the command in step 5.
7. Create a network load balancer for the new instance group created, and run the following command:

```
gcloud compute forwarding-rules create nginx-lb \
    --ports 80 \
    --target-pool nginx-pool
```

8. When prompted for a region, choose the region your instance group is in; for this example, you would choose us-east1.
9. Run the following command to allow HTTP access to your instance group:

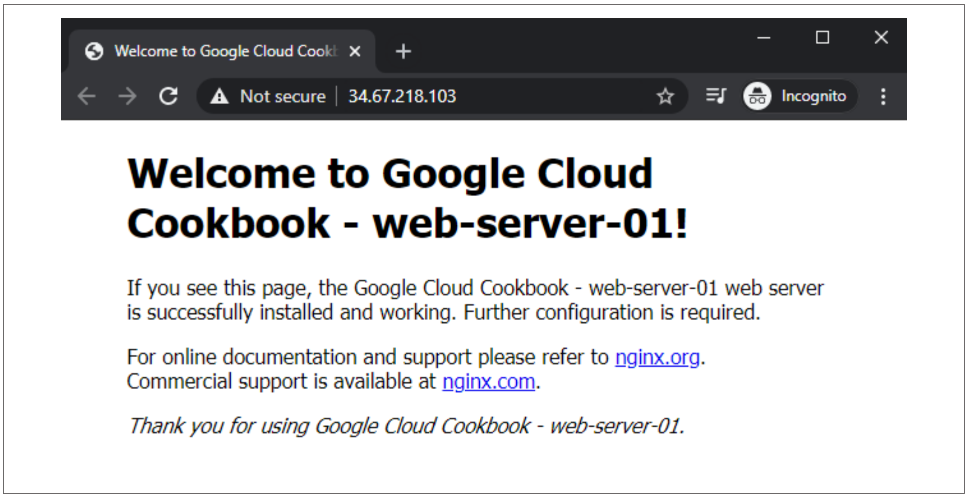
```
gcloud compute firewall-rules create allow-80 --allow tcp:80
```

10. To get the IP address of the regional load balancer, run the following command:

```
gcloud compute forwarding-rules list
```

11. Visit the associated IP address in your browser.

In your browser, you will see a screen that looks like Figure 5-22.



Welcome to Google Cloud Cookbook - web-server-01!

If you see this page, the Google Cloud Cookbook - web-server-01 web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using Google Cloud Cookbook - web-server-01.

Figure 5-22. NGINX running on Google Compute Engine

Discussion

In this recipe, you were introduced to managed instance groups and instance templates. Instance templates are a great method to deploy instances that require identical settings. An example is creating web servers for a cluster: you want all the instances to have the same configuration as memory, virtual CPUs, and the associated required software. You then used the instance template to create an instance group. An instance group is a grouping of virtual instances managed by a single entity. As a plus, you deployed a load balancer to distribute traffic to the virtual instances in the instance group. If one instance failed, your application would still be running, because you had additional instances in the instance group.

5.7 Deploying Containers to Managed Instance Groups

Problem

You have a requirement to start running your applications as containers. You want to get started with Compute Engine, and you want to run your NGINX as a container. You also want the benefits that Kubernetes provides, such as autoscaling, autohealing, and rolling updates.

Solution

Create a new Docker container. You will then deploy the Docker container to an instance group that will provide the autoscaling, autohealing, and rolling update requirements for your application. Instance groups provide you with benefits similar to Kubernetes, because it allows you to create MIGs to provide autoscaling, autohealing, and automatic updating.

1. Sign in to Google Cloud Console and launch Cloud Shell.
2. Create a new instance template and associate it with a publicly accessible container image. Run the following command in your Cloud Shell:

   ```
   gcloud compute instance-templates create-with-container nginx-template \
              --container-image gcr.io/cloud-marketplace/google/
   nginx1:1.15 \
              --tags http-server
   ```

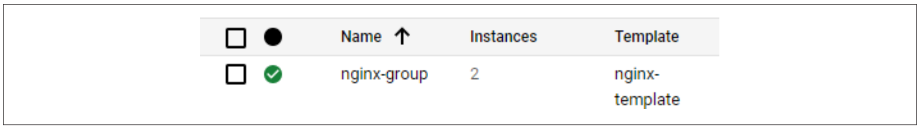
3. Create a target pool so you have a single access point for load balancing. Run the following command:

   ```
   gcloud compute target-pools create nginx-pool
   ```

4. Run the following command to create an instance group based on the newly created template:

   ```
   gcloud compute instance-groups managed create nginx-group \
       --base-instance-name nginx-vm \
       --size 2 \
       --template nginx-template \
       --target-pool nginx-pool
   ```

5. In the Cloud Console, navigate to your instance groups. You should see your instance group listed as shown in Figure 5-23.



☐	●	Name ↑	Instances	Template
☐	✓	nginx-group	2	nginx-template

Figure 5-23. Instance group virtual machines

6. Run the following command to create a regional load balancer for the newly created instance group:

```
gcloud compute forwarding-rules create nginx-lb \
    --ports 80 \
    --target-pool nginx-pool
```

7. When prompted for a region, choose the region your instance group is in; for this example, you would choose us-east1.

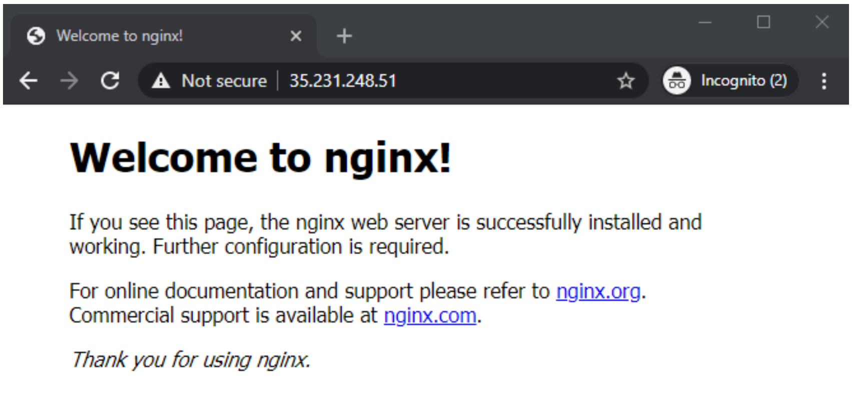
8. To get the IP address of the regional load balancer, run the following command:

```
gcloud compute forwarding-rules list
```

9. The output will print the IP address that you can copy and paste in a web browser.

10. Visit the associated IP address in your browser.

In your browser, you will see a screen that looks like Figure 5-24.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 5-24. NGINX running on Google Compute Engine

Discussion

In this recipe, you learned how to deploy a container to Compute Engine. This is a great way to get started with containers. It is recommended to run containers on Google Kubernetes Engine as the orchestration engine for your microservices. After deploying the containers, you exposed them with the `--tags http-server` flag, and you created a load balancer to distribute the traffic across the containers running on multiple virtual machines.

5.8 Transferring Files to Your Virtual Machine

Problem

You have a Linux virtual machine running on Compute Engine, and you need to transfer files to the instance.

Solution

In this recipe, you will use two methods of transferring files to your Linux virtual machine, one using the gcloud command-line tool and a second using SSH in your web browser.

1. You will need a Linux virtual machine running on Compute Engine to continue with this recipe.
2. Sign in to Google Cloud Console and launch Cloud Shell.
3. In your cloud shell, run the following command:

   ```
   touch myfile-one.txt
   ```

4. To copy the newly created file, run the following command and replace *-instance-name* with your instance name:

   ```
   gcloud compute scp myfile-one.txt instance-name:~
   ```

5. Connect to the Linux virtual machine with the Cloud Console SSH browser.
6. In the Linux terminal, validate the file copied by listing the files, using the **`ls`** command.
7. To upload a file with the SSH browser, click the Settings icon and select Upload File, as shown in Figure 5-25.

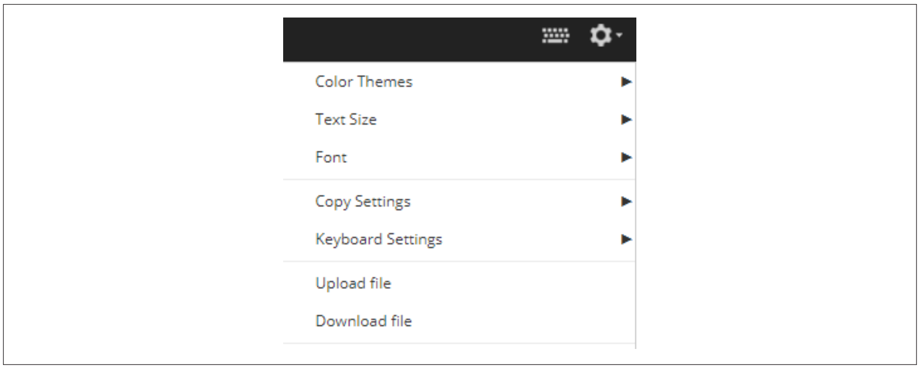
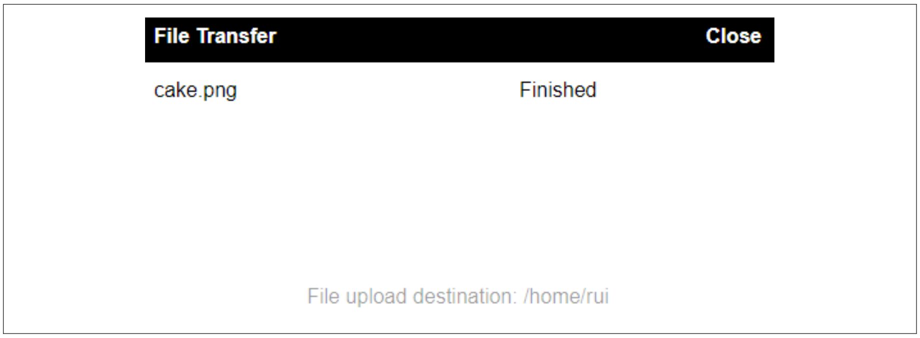


Figure 5-25. Menu for Settings, including the ability to upload and download files

8. Upload a sample file from your local workstation; Figure 5-26 shows the file transfer competition process.



cake.png Finished

File upload destination: /home/rui

Figure 5-26. File transfer completed

Discussion

Google Cloud provides multiple ways to transfer files to your virtual machines. In this recipe, you learned two methods to transfer files, one via the gcloud command that can be run on your local machine and the other via a web browser.

5.9 Using VM Manager for Patch Management

Problem

You host your virtual machines on Compute Engine, and you need a method for patching all the operating systems at once.

Solution

Using OS patch management with VM Manager, create a patch job to patch your fleet of Linux virtual machines.

1. You will need at least one Linux virtual machine running on Compute Engine to continue with this recipe.
2. Sign in to Google Cloud Console and launch Cloud Shell.
3. To have Compute Engine manage your operating systems, you will need to install and configure VM Manager and run the following commands to enable operating system management for all your virtual machines, replacing the PROJECT_ID with your Google Cloud project ID:

```
gcloud compute project-info add-metadata \
  --project PROJECT_ID \
  --metadata=enable-osconfig=TRUE

gcloud compute project-info add-metadata \
  --project PROJECT_ID \
--metadata=enable-guest-attributes=TRUE,enable-osconfig=TRUE
```

4. Using the Cloud Shell SSH browser, connect to one of your instances to validate that the OS Config Agent is installed, and run the following command:

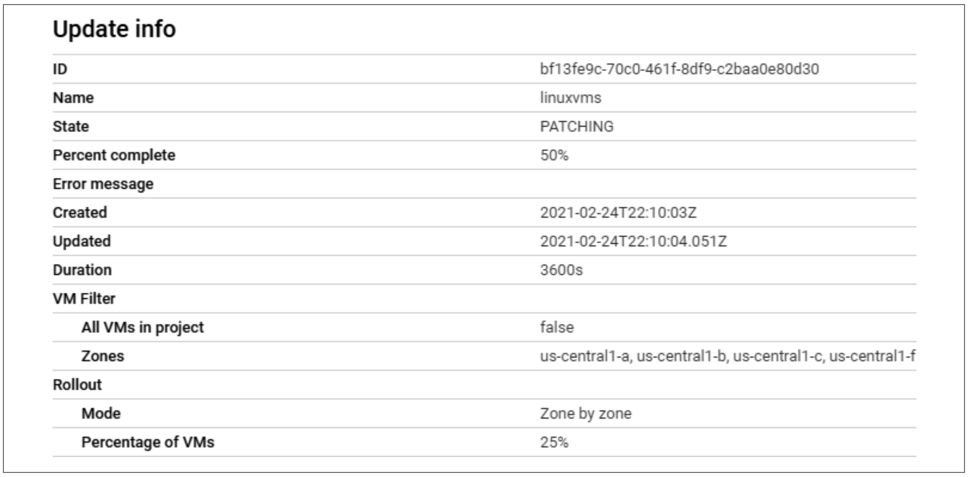
```
sudo systemctl status google-osconfig-agent
```

5. If the agent is running, you should see an output with the following statement:

```
active (running)
```

6. In the Google Cloud Console, navigate to Compute Engine > VM Manager > OS Patch Management.
7. Click Enable VM Manager.
8. Click New Patch Deployment.
9. Select the target zones for your virtual machines.
10. Click Next.
11. Enter a deployment name.
12. Click Next.

13. Choose default options for Scheduling, Rollout Options, and Advanced Options.
14. Click Deploy.
15. Your patch job will start automatically and display a status similar to Figure 5-27.



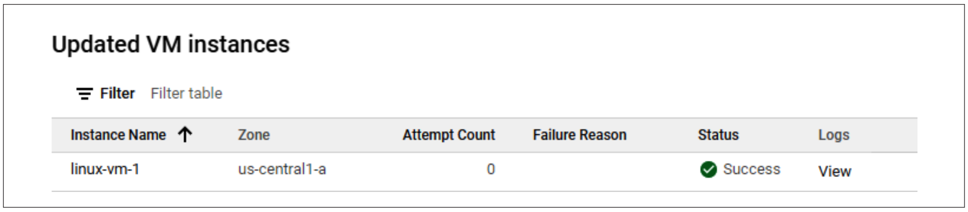
Update info

ID	bf13fe9c-70c0-461f-8df9-c2baa0e80d30
Name	linuxvms
State	PATCHING
Percent complete	50%
Error message	
Created	2021-02-24T22:10:03Z
Updated	2021-02-24T22:10:04.051Z
Duration	3600s
VM Filter	
All VMs in project	false
Zones	us-central1-a, us-central1-b, us-central1-c, us-central1-f
Rollout	
Mode	Zone by zone
Percentage of VMs	25%

Figure 5-27. Patching job status

Once the patch job is completed, you should see a status window, as shown in Figure 5-28.

OS patch management also supports Windows virtual machines.



Updated VM instances

Filter Filter table

Instance Name ↑	Zone	Attempt Count	Failure Reason	Status	Logs
linux-vm-1	us-central1-a	0		✓ Success	View

Figure 5-28. Patching job successfully completed

Discussion

When managing one virtual machine, it's fairly easy to perform patch updates. However, when running large fleets of virtual machines, it becomes difficult to patch all your instances without some complicated scripting or third-party tool. With VM Manager and OS patch management, Google Cloud provides the tools to distribute patch updates with multiple options, including the ability to do rolling updates and restrict updates to certain zones.

5.10 Backing Up Your Virtual Machine

Problem

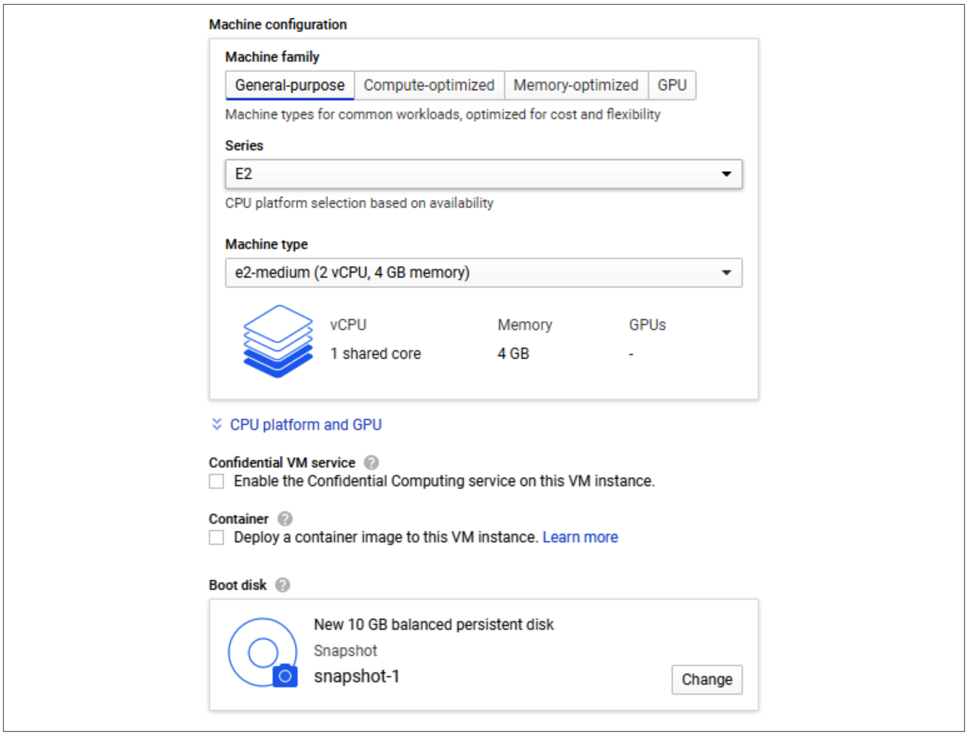
You have a Linux virtual machine running on Compute Engine. This is a critical application to your business, and you want to perform backups of the persistent disks assigned to the virtual machine as a method to recover the instance if something goes wrong.

Solution

Using Compute Engine persistent disk snapshots, you will create a snapshot of the virtual machine's persistent disk to have a recovery point of the disk in the event of mishap.

1. You will need a Linux virtual machine running on Compute Engine to continue with this recipe.
2. Sign in to Google Cloud Console.
3. Navigate to Compute > Compute Engine > Snapshots.
4. Click Create Snapshot.
5. Enter a name for the snapshot and select the source disk of your virtual machine.
6. Click Create.

Once completed, you should see your snapshot listed. You can now create a new virtual machine based on this snapshot. Figure 5-29 shows a virtual machine creation window with Boot Disk selected as the snapshot created.



Machine configuration

Machine family

General-purpose | Compute-optimized | Memory-optimized | GPU

Machine types for common workloads, optimized for cost and flexibility

Series

E2

CPU platform selection based on availability

Machine type

e2-medium (2 vCPU, 4 GB memory)

vCPU	Memory	GPUs
1 shared core	4 GB	-

CPU platform and GPU

Confidential VM service

Enable the Confidential Computing service on this VM instance.

Container

Deploy a container image to this VM instance. Learn more

Boot disk

New 10 GB balanced persistent disk

Snapshot

snapshot-1

Figure 5-29. Virtual machine creation with snapshot

Discussion

Using snapshots is a great method to protect your data in case something goes wrong. It creates a point-in-time copy of your persistent disk. Besides being a recovery point, it also allows you to create a virtual machine based on the snapshot, allowing you to perform file-level recovery and testing changes in your application or operating system. You can create a virtual machine with the disk that was snapshot, maybe to test how an operating system change would affect the state of the virtual machine.

CHAPTER 6

Google Cloud Kubernetes Engine

Google Cloud Kubernetes Engine (GKE) is a fully managed and secured platform that provides you the ability to run your containerized workloads. This chapter contains recipes for creating and managing your containers, including unique methods to automate deployments and ways to deploy real-world applications.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.

6.1 Creating a Zonal Cluster

Problem

You want to run an application on Kubernetes but want to run it only within a single zone, within a region. You also want to be able to create and upgrade your Kubernetes cluster quickly.

Solution

Run your application on a zonal Kubernetes cluster. With a single control plane managing your Kubernetes cluster, it's very easy to get started quickly.

Prerequisites

Ensure that the Kubernetes Engine API is enabled.

1. Sign in to the Google Cloud Console.
2. In the main menu, navigate to Compute and click Kubernetes Engine.
3. Click the Create button at the top of the screen.
4. Click Configure next to the Standard option.
5. In the Cluster Basics section:
   a. Choose a name for your cluster.
   b. In Location Type, select Zonal.
   c. In Zone, select any zone of your choice.
   d. Leave the remaining settings at the defaults.
6. In the left navigation pane, several other options could be set; however, we will leave them at the defaults for the purposes of this recipe.
7. Click Create at the bottom of the screen.
8. You will be navigated back to the Clusters screen, where you will see your cluster spinning up. This process can take more than a minute to complete.
9. Once complete, you will see a green checkmark icon next to the name of your cluster. Your cluster is now ready for the deployment of applications.

Discussion

Creating a zonal GKE cluster is a quick and easy way to get going with Kubernetes versus trying to launch a self-managed Kubernetes cluster. GCP manages the Kubernetes control plane, so you don't need to worry about the operational overhead that comes with managing it. Beyond this recipe, you should look through the configuration options you have with GKE, which affords you tons of flexibility and additional configuration around nodepools, automation, networking, security, metadata, and more. For additional information, refer to the Google Kubernetes Engine guide.

6.2 Creating a Regional Cluster

Problem

You want to run an application on Kubernetes, but across two or more zones within a region. You value the availability of your application over the flexibility that may come with a zonal Kubernetes cluster.

Solution

Run your application on a regional Kubernetes cluster. Regional clusters allow for higher availability, fault tolerance, and no-downtime upgrades. This makes your application more resilient and spread across multiple zones within a single region.

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Kubernetes Engine.
3. Click the Create button at the top of the screen.
4. Click Configure next to the Standard option.
5. In the Cluster Basics section:
    a. Choose a name for your cluster.
    b. In Location Type, select Regional.
    c. In Region, select any region of your choice.
    d. Leave the remaining settings at their defaults.
6. In the left navigation pane, several other options could be set; however, we will leave them at their defaults for the purposes of this recipe.
7. Click Create at the bottom of the screen.
8. You will be navigated back to the Clusters screen, where you will see your cluster spinning up. This process can take more than a minute to complete.

Once complete, you will see a green checkmark icon next to the name of your cluster.

Discussion

Creating a regional GKE cluster is a quick and easy way to get going with Kubernetes versus trying to self-manage a Kubernetes cluster of your own. With a regional cluster, you have nodes deployed across the zones within that region, so expect that the number of nodes, total vCPUs, and total memory are larger than your zonal GKE deployment with the same configuration. The Kubernetes control plane (managed by GCP) is also spread out across the zones within the region, so you don't need to worry about configuring it beyond deploying the cluster itself. Beyond this recipe, you should look through the configuration options you have with GKE, which afford tons of flexibility and configurability around nodepools, automation, networking, security, metadata, and more.

Summary of the benefits of running a regional GKE cluster:

- Resilience from single zone failure
- Continuous control plane upgrades

6.3 Resizing a Cluster

Problem

You are running a Kubernetes cluster that has either too few nodes (and therefore is unable to meet spiking demand for your application) or a Kubernetes cluster that has too many nodes (and is over-provisioned for the level of traffic it is receiving), and you want to resize your cluster.

Solution

You should resize the number of nodes your Kubernetes cluster is running to ensure that you have set up an optimal cluster based on your application's traffic patterns.

Prerequisites

Ensure that the Kubernetes Engine API is enabled as well as that you have a zonal or regional cluster running that you can resize. (See Recipes 6.1 and 6.2.)

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Kubernetes Engine.
3. If you completed the preceding recipes, you should have at least one Kubernetes cluster running. In Figure 6-1, you will see we have two regional clusters running, one in us-east1 and another in us-west1. For this recipe, we will increase the number of nodes in cluster-1.

> The process is the same for resizing a cluster's nodes, whether it's regional or zonal.

4. Click the name of your cluster.
5. Click Nodes underneath the name of your cluster, as shown in Figure 6-1.

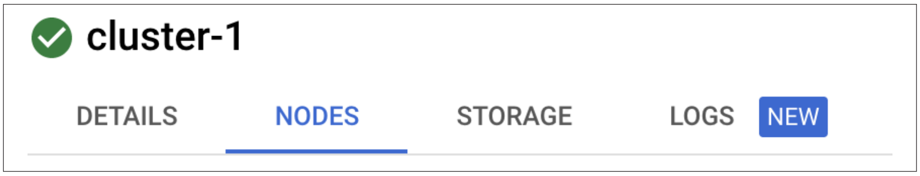
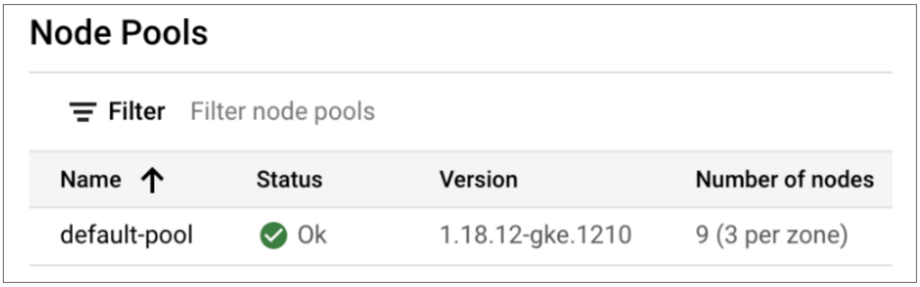


Figure 6-1. Nodes tab within selected cluster

6. You should now see one node pool, called default-pool. Click default-pool (or whatever the name of your particular node pool is), as shown in Figure 6-2.



Node Pools

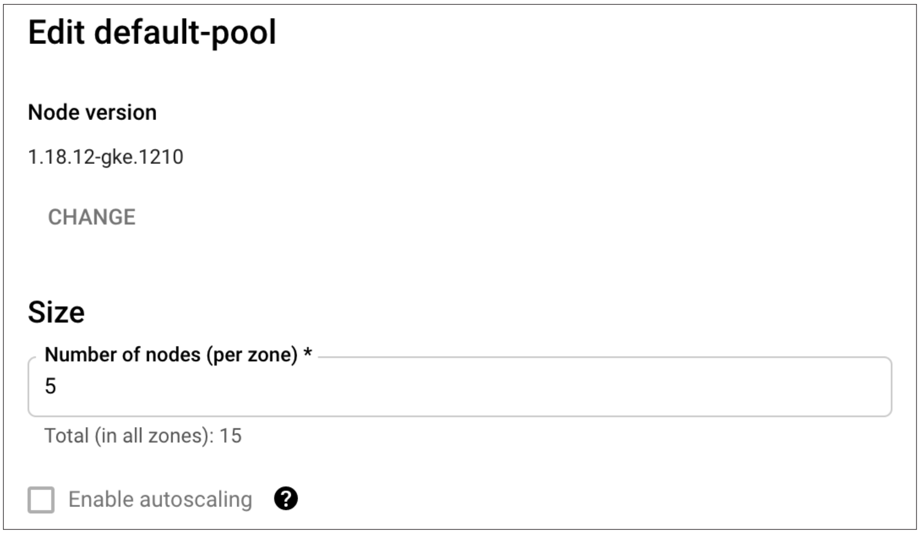
Filter Filter node pools

Name ↑	Status	Version	Number of nodes
default-pool	✓ Ok	1.18.12-gke.1210	9 (3 per zone)

Figure 6-2. Node pools

7. Click Edit at the top of the Node Pools screen.
8. Now we can increase and decrease the default size of our node pool to any number of nodes that we prefer, as shown in Figure 6-3. In Figure 6-3, we will increase the node pool size from 3 to 5.

If you want your Kubernetes cluster to autoscale up based on node utilization, select the Enable Autoscaling box. Selecting the box will give you the option to set minimum and maximum node thresholds.


Edit default-pool

Node version

1.18.12-gke.1210

Size

Number of nodes (per zone) *

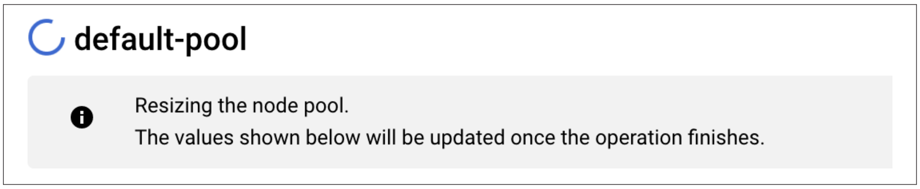
5

Total (in all zones): 15

☐ Enable autoscaling

Figure 6-3. Increase nodes in the default-pool

9. Click the Save button at the bottom of the screen. You should see the screen shown in Figure 6-4.


default-pool

Resizing the node pool.
The values shown below will be updated once the operation finishes.

Figure 6-4. Resize node pool

This may take a minute or two, but once completed, you have effectively resized your default node pool from running three nodes to running five nodes in the default-pool node pool.

Discussion

Resizing the number of nodes in a node pool is a relatively simple process. The number of nodes you run, as well as the number of node pools you run, should be thought out carefully to meet the needs of your particular application. Enabling autoscaling in your node pools is a major value add that Kubernetes brings allowing your node pools to increase and decrease based on the utilization of that node within a range you specify.

6.4 Automatically Routing Traffic to the Nearest Cluster with Multi-Cluster Ingress

Problem

You have an application that runs on multiple Kubernetes clusters that are located in different regions, and you want to be able to route user traffic automatically to the cluster that is nearest to the user's location, using a single HTTP(S) load balancer.

Solution

Use Multi-Cluster Ingress for Anthos to run your application across as many Kubernetes clusters as you'd like, and route traffic to the nearest cluster, based on the origin of the request.

Prerequisites

Ensure that the following APIs are enabled:

- Kubernetes Engine API
- GKE Hub
- Anthos
- Multi-Cluster Ingress API

First, we will create two regional clusters in two regions (us-east1 and us-west1).

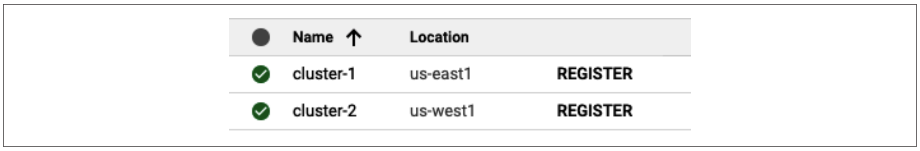
1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Kubernetes Engine.
3. Click the Create button at the top of the screen.
4. Click Configure next to the Standard option.
5. In the Cluster Basics section:
   a. In Name, set your cluster name to cluster-1.
   b. In Location Type, select Regional.
   c. In Region, select us-east1.
   d. Leave the remaining settings at their defaults.
6. Click Create at the bottom of the screen.

   You will be navigated back to the Clusters screen, where you will see your cluster spinning up. This process can take more than a minute to complete.

7. We will repeat this process and create another regional cluster in a different region than the one we just created.
8. Click the Create button at the top of the screen.
9. Click Configure next to the Standard option.
10. In the Cluster Basics section:
    a. In Name, set your cluster name to cluster-2.
    b. In Location Type, select Regional.
    c. In Region, select us-west1.
    d. Leave the remaining settings at their defaults.
11. Click Create at the bottom of the screen. You will be navigated back to the Clusters screen, where you will see your cluster spinning up. This process can take more than a minute to complete.

    Now we will register the clusters to the same environment.
12. In the main menu, navigate to Anthos and click Clusters in the submenu.
13. Click Register Existing Cluster.

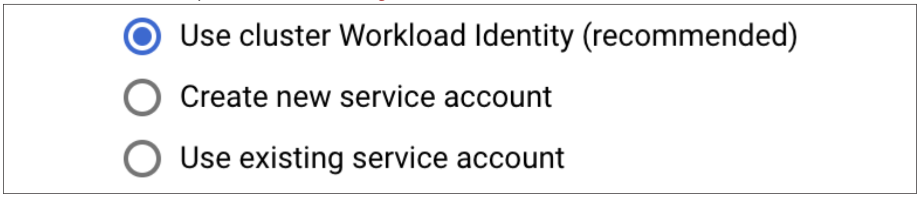
    You will now see that both the clusters you created are ready to be registered as shown in Figure 6-5.



●	Name ↑	Location	
✓	cluster-1	us-east1	REGISTER
✓	cluster-2	us-west1	REGISTER

Figure 6-5. Two clusters running in the Google Cloud Console

14. Next to cluster-1, click REGISTER.
15. You'll be asked for a service account to register to the environment; choose Workload Identity as shown in Figure 6-6.



[✓ Use cluster Workload Identity (recommended)]
Create new service account
Use existing service account

Figure 6-6. Register service account

16. Click Submit.
17. Repeat steps 13-15 for the second cluster, cluster-2.

Now, we will set up Ingress for Anthos.

18. Next, click Features in the Anthos screen.
19. Click Enable next to Ingress and then click Enable Ingress.
20. In the Config Membership drop-down menu, select the first cluster you spun up (cluster-1) and click Install. After a minute or so, refresh the screen, and you should see the Ingress Enabled screen.
21. Open the cloud shell by clicking this button in the top-right corner of your screen.
22. Type the following into your cloud shell to make a directory that will hold the *.yaml* files we will need for the remainder of this tutorial:

    ```
    mkdir multicluster-ingress-demo \
        && cd multicluster-ingress-demo
    ```

23. Before we can work with our clusters via kubectl in the cloud shell, we need to configure our cluster access by generating a kubeconfig entry. You can do this by running the following command for both of your clusters in the cloud shell:

    ```
    gcloud container clusters \
        get-credentials cluster-1 --region us-east1
    gcloud container clusters \
         get-credentials cluster-2 --region us-west1
    ```

24. Ensure that you received a confirmation for each cluster:

    ```
    Fetching cluster endpoint and auth data.
    kubeconfig entry generated for cluster-1.
    Fetching cluster endpoint and auth data.
    kubeconfig entry generated for cluster-2
    ```

25. Now we can work with the clusters from the cloud shell command line. Let's create the namespace for our application to run. You can do this by typing **`nano namespace.yaml`** in the cloud shell.

    Paste this into the *.yaml* file:

    ```
    apiVersion: v1
    kind: Namespace
    metadata:
      name: zoneprinter
    ```

26. Save the file. Before we proceed, let's set the shell variable for our project ID. Enter the following in the cloud shell:

    ```
    PROJECT=$(gcloud info --format='value(config.project)')
    ```

27. Now let's apply *namespace.yaml* to both of our clusters, cluster-1 and cluster-2. You can do this by running the following:

    ```
    kubectl config use-context \
        gke_$(echo $PROJECT)_us-east1_cluster-1
    kubectl apply -f namespace.yaml
    kubectl config use-context \
    ```

```
        gke_$(echo $PROJECT)_us-west1_cluster-2
    kubectl apply -f namespace.yaml
```

We will now deploy a sample app, which shows the location of the data center you are reaching to both clusters, from an image called zone-printer.

28. Create a new *.yaml* file by typing **`nano app.yaml`** in the gcloud terminal, and paste the following into the yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: zone-ingress
  namespace: zoneprinter
  labels:
    app: zoneprinter
spec:
  selector:
    matchLabels:
      app: zoneprinter
  template:
    metadata:
      labels:
        app: zoneprinter
    spec:
      containers:
      - name: frontend
        image: gcr.io/google-samples/zone-printer:0.2
        ports:
        - containerPort: 8080
```

29. Save the file. Apply *app.yaml* to both of your clusters, cluster-1 and cluster-2. You can do this by running the following in the cloud shell:

```
kubectl config use-context \
gke_$(echo $PROJECT)_us-east1_cluster-1
kubectl apply -f app.yaml
kubectl config use-context \
gke_$(echo $PROJECT)_us-west1_cluster-2
kubectl apply -f app.yaml
```

Now that the app is running in both clusters in the same namespace, let's wrap up by creating the MultiClusterService and MultiClusterObject.

30. First, create the MultiClusterService. Create a new *.yaml* file by typing **`nano mcs.yaml`** in the cloud shell, and paste the following into the YAML:

```
apiVersion: networking.gke.io/v1
kind: MultiClusterService
metadata:
  name: zone-mcs
  namespace: zoneprinter
spec:
  template:
    spec:
      selector:
        app: zoneprinter
      ports:
      - name: web
        protocol: TCP
        port: 8080
        targetPort: 8080
```

31. Save the file and apply this file to cluster-1:

    ```
    kubectl config use-context \
              gke_$(echo $PROJECT)_us-east1_cluster-1
    kubectl apply -f mcs.yaml
    ```

32. Create the MultiClusterIngress. Create a new *.yaml* file by typing **`nano mci.yaml`** in the cloud shell, and paste the following into the YAML:

    ```
    apiVersion: networking.gke.io/v1
    kind: MultiClusterIngress
    metadata:
      name: zone-ingress
      namespace: zoneprinter
    spec:
      template:
        spec:
          backend:
            serviceName: zone-mcs
            servicePort: 8080
    ```

33. Save the file and apply this file to cluster-1:

    ```
    kubectl apply -f mci.yaml
    ```

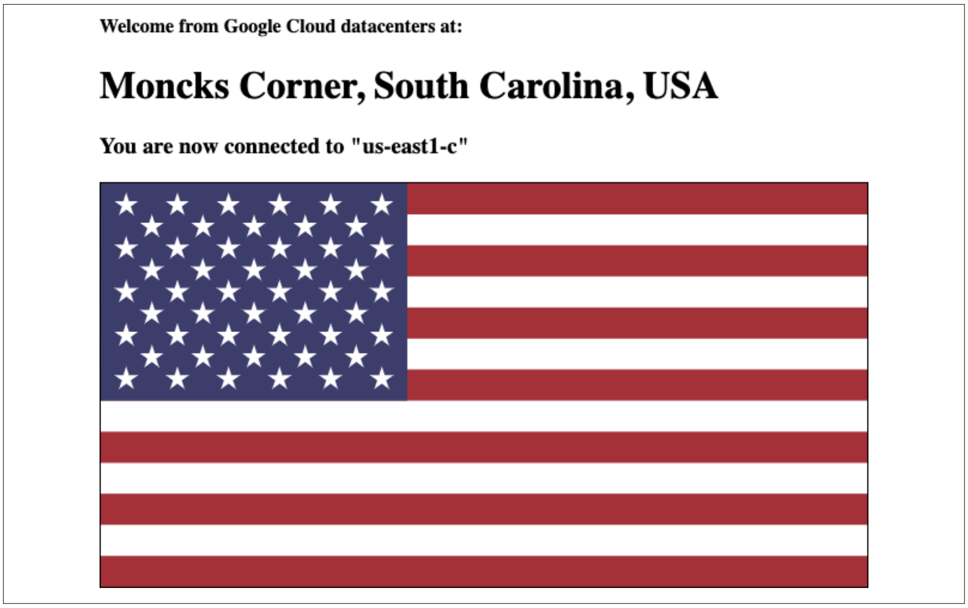
34. Finally, pull the virtual IP (VIP) to access our application from the MultiCluster Ingress. Run this command in your cloud shell:

    ```
    kubectl describe mci zone-ingress -n zoneprinter
    ```

35. In the output, under the Status heading, you will find an entry that says VIP: <ip address>. If you don't see VIP: <ip address> immediately, that's OK; the ingress may take a few minutes to spin up. Keep running the **`describe`** command until you see the IP appear.

36. Once you get the VIP, open a new tab, paste the VIP to the URL bar, and press Enter.

    You should see a web page, as shown in Figure 6-7.


Welcome from Google Cloud datacenters at:

Moncks Corner, South Carolina, USA

You are now connected to "us-east1-c"

Figure 6-7. Running application on multi-cluster ingress

Discussion

In summary and in order: we created two GKE clusters in us-east1 and us-west1 and then registered the clusters to an environment and enabled the Ingress for Anthos feature. We then created the proper namespace and deployed the zone-printer application to both clusters, used cluster-1 as our config cluster, and deployed a multi-cluster service and Multi-Cluster Ingress to that cluster. The request now routes through an L7 HTTP load balancer to the nearest cluster running the application from the location of the request. Multi-Cluster Ingress, using Ingress for Anthos, will allow you to route requests to your Kubernetes clusters running anywhere in the world.

6.5 Deploying a Spring Boot Java Application

Problem

You need to deploy a Java Spring Boot REST service to Kubernetes.

Solution

Use Google Cloud Source Repositories, Google Cloud Container Builder, and Jib to containerize and deploy the Spring Boot REST service to a Kubernetes cluster.

For this recipe, you will need to `git clone` this book's code example repository.

1. On your local workstation, open the working directory for this recipe from the cloned repository:

   ```
   cd google-cloud-cookbook/06-kubernetes/6-8-java
   ```

2. Test the sample Java application locally by running the following command:

   ```
   ./mvnw -DskipTests spring-boot:run
   ```

3. In your browser, go to *http://localhost:8080*. You should see a screen similar to Figure 6-8.

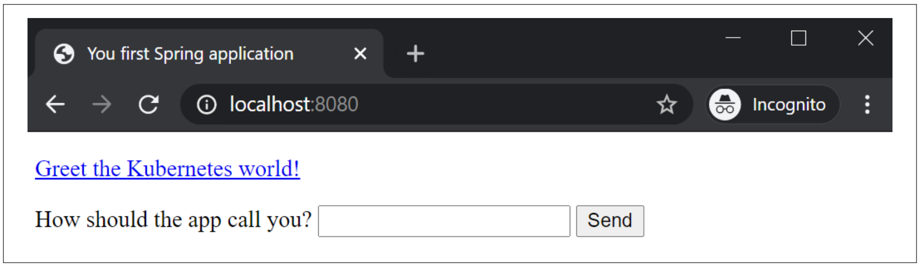

You first Spring application
localhost:8080
Incognito
Greet the Kubernetes world!
How should the app call you?
Send

Figure 6-8. Java Spring Boot application

4. Run the following command to enable the Google Cloud Container Registry API to store the container image:

   ```
   gcloud services enable containerregistry.googleapis.com
   ```

5. Use Jib to create the container image and push it to the Container Registry; replace $GOOGLE_CLOUD_PROJECT with your Google Cloud Project ID:

   ```
   mvn compile \
       com.google.cloud.tools:jib-maven-plugin:2.0.0:build \
       -Dimage=gcr.io/$GOOGLE_CLOUD_PROJECT/hello-java:v1
   ```

6. Build and push the image to a container registry:

   ```
   mvn compile jib:build \
       -Dimage=gcr.io/ruicosta-blog/hello-java:v1
   ```

7. To test the Docker installation, run the following command:

   ```
   mvn compile jib:dockerBuild \
       -Dimage=gcr.io/ruicosta-blog/hello-java:v1
   ```

8. To list the Docker images, run the following command:

   ```
   docker images
   ```

9. Run the following command to run the Docker container locally on your machine, and replace the image ID with yours from step 8:

   ```
   docker run -p 8080:8080 -t IMAGE_ID
   ```

10. In your browser, go to *http://localhost:8080*, and you should see a similar screen to the one in step 3. You have now tested the Spring Boot application, containerized locally.

11. Create a Kubernetes two-node cluster:

    ```
    gcloud container clusters create hello-java-cluster \
        --num-nodes 2 \
        --machine-type n1-standard-1 \
        --zone us-central1-c
    ```

12. To deploy your application to the GKE cluster, run the following command and replace GOOGLE_CLOUD_PROJECT with your Google Cloud project ID:

    ```
    kubectl create deployment hello-java \
        --image=gcr.io/$GOOGLE_CLOUD_PROJECT/hello-java:v1
    ```

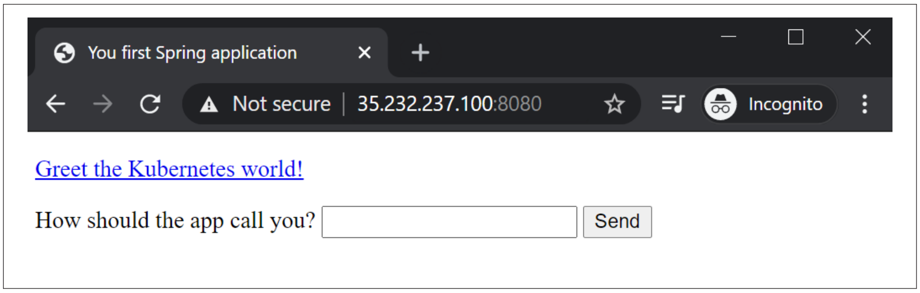
13. To make the hello-java container accessible from outside the GKE cluster, you will have to expose the pod as a Kubernetes service. Run the following command:

    ```
    kubectl create service loadbalancer hello-java --tcp=8080:8080
    ```

14. To find the publicly accessible IP address of the service, run the following command:

    ```
    kubectl get services
    ```

15. Visit *http://EXTERNAL-IP:8080* in your web browser; you should see a page similar to the one shown in Figure 6-9.


You first Spring application

Not secure | 35.232.237.100:8080

Greet the Kubernetes world!

How should the app call you?

Figure 6-9. Java Spring Boot application running on Kubernetes

Discussion

In this recipe, you deployed a Java Spring Boot application to Google Cloud Kubernetes Engine. You leveraged Jib, which builds containers without having to declare a Dockerfile. Jib was developed by Google to simplify the process of building Java containers—no need to create a Docker file or wait for the build to complete. Jib handles all the steps required to build your Java container.

6.6 Deploying a Java Application to Kubernetes, Using Skaffold

Problem

You need a method to develop, build, push, and deploy your Java application quickly to Kubernetes.

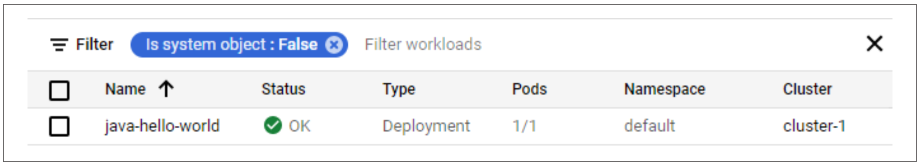
Solution

Use Skaffold and the Cloud Code plug-in for IntelliJ to develop, build, push, and deploy your application to Kubernetes, all from the IntelliJ IDE.

1. Install Cloud Code for IntelliJ.
2. Create a new IntelliJ project.
3. Choose Cloud Code: Kubernetes > Java: Hello World and click Next.
4. Enter the location of your container repository, as in the example shown here:

   ```
   gcr.io/ruicosta-blog
   ```
5. Choose a project name and location for your project files.
6. Navigate to the Kubernetes Explorer from the right-side panel or by going to Tools > Cloud Code > Kubernetes > View Cluster Explorer.
7. Select Add A New GKE Cluster and click Create A New GKE Cluster. This will open the Google Cloud Console in a web browser to the cluster wizard page.
8. In the Google Cloud Console, create a new cluster.

   Once your cluster is created, your screen should update with the cluster name.
9. Click OK.
10. Click Run On Kubernetes.
11. Once the process is complete, you should see the workload created in the Google Cloud Console, as shown in Figure 6-10.



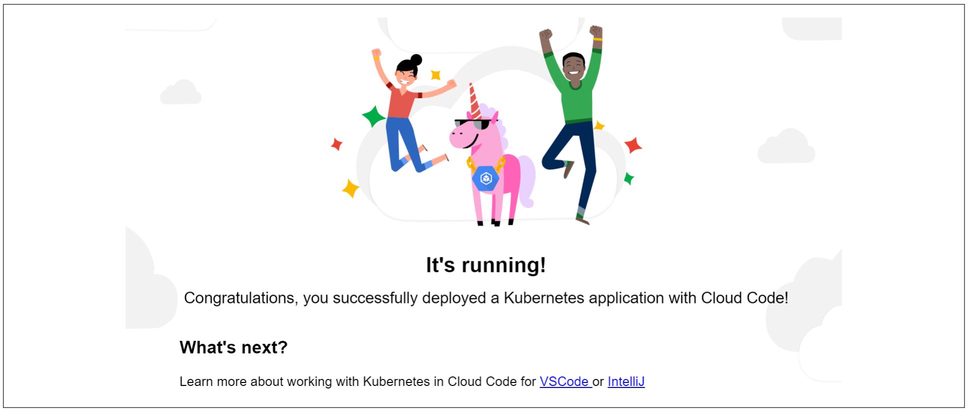
☐	Name ↑	Status	Type	Pods	Namespace	Cluster
☐	java-hello-world	✓ OK	Deployment	1/1	default	cluster-1

Figure 6-10. Kubernetes workload

Besides making it easy to deploy your application from IntelliJ to Google Cloud, it also tunnels the traffic from your local workstation to Kubernetes.

12. In your web browser, go to *http://localhost* to see a screen similar to Figure 6-11.

The application is now running on a Google Cloud Kubernetes cluster in your Google Cloud project.



It's running!

Congratulations, you successfully deployed a Kubernetes application with Cloud Code!

What's next?

Learn more about working with Kubernetes in Cloud Code for VSCode or IntelliJ

Figure 6-11. Java application running on Kubernetes

Discussion

Google Cloud Code is an amazing add-on for your IntelliJ or Visual Studio code editors to deploy and debug your code faster. With Google Cloud Code, you can easily enable Google Cloud APIs, create clusters, and deploy your applications to either Cloud Run or Kubernetes. Google Cloud Code leverages Skaffold, which is a tool that handles the pipeline for building, pushing, and deploying your application to Kubernetes. For additional information, please refer to documentation for Skaffold and Google Cloud Code.

6.7 Using GKE Autopilot for Running an Application You Don't Have to Manage

Problem

You want to run your application on Kubernetes, but don't want to have to manage nodes, node pools, images, networking, and the other operational components of running a Kubernetes cluster. Effectively, you want to run your application(s) on Kubernetes while not having to worry about the management or operation of the cluster itself.

Solution

Run your application on GKE Autopilot. With GKE Autopilot, many operational aspects of Kubernetes are abstracted away, and you are left with a Kubernetes infrastructure that is largely configured to Google GKE best practices.

1. Sign in to Google Cloud Console.
2. In the main menu, navigate to Compute and click Kubernetes Engine.
3. Click the Create button at the top of the screen.
4. Click Configure next to the Autopilot option.
5. In the Cluster Basics section:
   a. In Name, give your GKE Autopilot cluster any name of your choice.
   b. In Region, pick any region of your choice.
   c. The remaining options can be left at their defaults. Click the Create button at the bottom of the screen.
   d. The cluster should take a minute or so to spin up and be ready for use.

Discussion

At the time of writing, GKE Autopilot is a new GCP offering that allows users to run Kubernetes clusters in a more managed format versus running GKE in Standard mode. The managed aspect of GKE Autopilot reduces the users' full control over the cluster in exchange for an ease of operational overhead. GKE Autopilot is an exciting mode of operation that enables users to get started much more quickly deploying their production workloads in GCP.

CHAPTER 7

Working with Data

One of the greatest paradigm shifts when working with cloud computing is the nearly unlimited storage now available to users. Cheap, scalable blob storage in the form of Google Cloud Storage (GCS) allows administrators to start from a standpoint of "never delete data." Services like BigQuery and Spark on Dataproc allow you to pay for long-lived storage separately from the compute resources, which you pay for by the second. Generally, compute is more expensive than storage, so this paradigm saves on a great deal of engineering effort trying to move, archive, and retrieve data between disparate storage systems.

The recipes in this chapter show tips and tricks when working with the various data layers of Google Cloud, from moving data round GCS buckets faster, to automatically archiving long-term data, to some more advanced database techniques.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

7.1 Speeding Up Cloud Storage Bulk Transfers by Multiprocessing

Problem

Although the gsutil tool performs well and is a great CLI solution for interacting with GCS, sometimes you want to max out your CPU and network bandwidth for a faster transfer. You'll often do this when transferring a large number of files, either to or from GCS or within the GCS service.

Solution

You can leverage the -m flag when using the gsutil command-line tool to multiprocess your transfer.

1. From a CLI with gcloud and gsutil installed or from Cloud Shell, create a bucket and upload data to observe normal single-process transfer speed:

    ```
    BUCKET_NAME=my-bucket-4312
    gsutil mb -l us gs://$BUCKET_NAME
    ```

2. Upload some data from a public bucket to your new bucket. This will move ~250 GB, so once you have reached a steady-state transfer speed, you can cancel the job. If you would like to save time, use the Ctrl+C key command:

    ```
    gsutil cp -r gs://gcp-public-data-landsat/LC08/01/044/034/* gs://
    $BUCKET_NAME
    ```

3. Add `-m` to multiprocess your transfer and observe greatly increased speed. When you see a steady-state speed, you can cancel the command:

    ```
    gsutil -m cp -r gs://gcp-public-data-landsat/LC08/01/044/034/* gs://
    $BUCKET_NAME
    ```

4. Delete test files to avoid long-term charges.

    ```
    gsutil -m rm -r  gs://$BUCKET_NAME/034/*
    ```

Discussion

Adding `-m` to the gsutil causes most commands to run in parallel, using a combination of multithreading and multiprocessing. The number of threads and processes are set by `parallel_thread_count` and `parallel_process_count`. These can be set in your *.boto configuration* file or set on the command line with the `-o` option flag.

In general, if you are moving a couple of files, you won't need this flag. However, if you are doing batch uploads and are OK saturating your CPU and even your network link, you can consider setting it. Just note that this can starve other processes or devices on the network of resources.

If you are trying to do this in code, it is usually more performant to use the appropriate client library rather than call the gsutil command-line tool, which is in turn just calling a Python library.

7.2 Speeding Up GCS Transfers for Large Files with Parallel Composite Uploads

Problem

You want to increase the speed for a file transfer, particularly to and from GCS for large files in the gigabyte range or above. The previous recipe covered transfers with a large number of operations; this one is targeted at individual large files.

Solution

Leverage parallel composite uploads, set at the command line or in your *.boto* configuration file for gsutil.

1. Create a bucket and download a single file to then perform test uploads:

   ```
   BUCKET_NAME=my-bucket-4312

   gsutil mb -l us gs://$BUCKET_NAME
   # copy a largish file locally (~250MB)
   gsutil cp gs://gcp-public-data-
   landsat/LC08/01/044/017/LC08_L1GT_044017_20200809_20200809_01_RT/
   LC08_L1GT_044017_20200809_20200809_01_RT
   _B8.TIF.
   ```

2. Upload the file as a single chunk and observe transfer speed:

   ```
   gsutil cp LC08_L1GT_044017_20200809_20200809_01_RT_B8.TIF gs://
   $BUCKET_NAME
   ```

3. Upload the file as several simultaneous chunks and observe transfer speed. If your previous upload saturated your link, you may not see a performance increase.

   You'll notice we set the file size threshold that triggers parallel composite uploads (250 MB) and how large these composite files should be (50 MB), and we tell gsutil to use eight processes to enable uploading these files at the same time. All of these can be tuned to increase performance.

   ```
   gsutil -o
   "GSUtil:parallel_composite_upload_threshold=200M,GSUtil
   :parallel_composite_upload_component_size=50M,GSUtil:parallel_pro-
   cess_count=8" cp LC08_L1GT_044017_20200809_20200809_01_RT_B8.TIF gs://
   $BUCKET_NAME
   ```

4. Delete the file to save on charges:

   ```
   gsutil rm gs://$BUCKET_NAME/
   LC08_L1GT_044017_20200809_20200809_01_RT_B8.TIF
   ```

Discussion

Leveraging parallel composite uploads is particularly helpful when uploading large files. However, this requires both the source and destination environments to have a CRC32C library installed for integrity checking. There are additional caveats, such as a maximum of 32 objects per composite. Take a look at Integrity Checking documentation for more information.

7.3 Mounting GCS as a Filesystem

Problem

You want to use traditional, filesystem-based tools to interact with GCS blobs and directories. Although you can easily use the client libraries or gsutil to access GCS, sometimes you'll encounter legacy applications that expect files to be accessible as a POSIX-compliant mount.

Solution

Use gcsfuse, a community-supported, open source convenience option, to mount a GCS bucket to your VM. These are the Linux instructions, although gcsfuse is also available for macOS:

1. Create a test VM, using your project ID and the latest Ubuntu image:

```
PROJECT_ID=<INSERT PROJECT>
gcloud compute --project=$PROJECT_ID instances create gcs-fuse-vm \
    --zone=us-central1-a --machine-type=e2-medium \
    --scopes=https://www.googleapis.com/auth/cloud-platform \
    --image=ubuntu-2004-focal-v20210315 --image-project=ubuntu-os
      -cloud \
    --boot-disk-size=100GB
```

2. Create your bucket and populate some test data:

```
BUCKET_NAME=<INSERT BUCKET>

gsutil mb -l us gs://$BUCKET_NAME
gsutil -m cp -r gs://gcp-public-data-landsat/LC08/01/044/034/
LC08_L1GT_044034_20130330_20170310_01_T2 gs://$BUCKET_NAME
```

3. SSH onto the VM.

4. Install FUSE Installer info (Ubuntu/Debian latest releases):

```
export GCSFUSE_REPO=gcsfuse-`lsb_release -c -s`
echo "deb http://packages.cloud.google.com/apt $GCSFUSE_REPO main" |
sudo tee /etc/apt/sources.list.d/gcsfuse.list
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-
key add -
```

5. Install gcsfuse:

```
sudo apt-get update
sudo apt-get install gcsfuse
```

6. Check the current gcloud credentials to see what gcsfuse will use:

```
gcloud auth list
```

7. Create a mount point and mount:

```
BUCKET_NAME=my-bucket-4312

mkdir $BUCKET_NAME
gcsfuse --implicit-dirs $BUCKET_NAME $BUCKET_NAME
```

8. View the files you previously uploaded to the bucket as if they were on the filesystem:

```
ls $BUCKET_NAME
```

9. Add a file and test that it's indeed on GCS:

```
cd $BUCKET_NAME
echo "gcs fuse is cool!" > file.txt

cd
gsutil cat gs://$BUCKET_NAME/file.txt
```

10. Double-check your user's ID and GID so you can mount on non-root (and note these for the following step):

```
id
```

11. Add an entry to */etc/fstab* to mount every time automatically when the VM boots using your UID and GID as well as bucket name:

```
sudo vim /etc/fstab

# insert:
my-bucket-4312 /home/dhodun/my-bucket-4312 gcsfuse
rw,implicit_dirs,_netdev,allow_other,uid=1002,gid=1003
```

12. Restart the VM and test to ensure that the bucket is always mounted at bootup:

```
sudo shutdown -r now

# on restart
ls my-bucket-4312
```

13. Delete the VM:

```
gcloud compute instances delete gcs-fuse-vm
```

Discussion

Gcsfuse is a great convenience tool for mounting and accessing GCS blobs. It is not a production replacement for a traditional file store—for that, consider Filestore—and lacks many features of a true POSIX filesystem as well as some GCS features such as metadata. You might use it to save on cost (Filestore is about 10x the price per gigabyte) or because you are trying to access data that is largely used and generated by systems compatible with GCS and you need a one-off workaround. Take a look at the mounting docs for more info on permissions and security.

Also note that you will still incur GCS storage charges, particularly for nearline and coldline (archival) storage (see FUSE notes).

7.4 Automatically Archiving and Deleting GCS Objects

Problem

You want to handle lifecycle management of GCS objects automatically, namely, changing the storage class to more archival-friendly classes as files get older and deleting the oldest files according to a policy.

Solution

You can leverage lifecycle management on a GCS bucket level with policies based on an object's age (or other attributes) to change storage class, delete, and other actions.

1. Create a bucket to be managed:

   ```
   BUCKET_NAME=my-bucket-4312
   gsutil mb -l us gs://$BUCKET_NAME
   ```

2. Create a *lifecycle_management.json* file with rules based on your business needs. This policy will then be applied to your bucket. The example policy will archive standard and Durable Reduced Availability (DRA) storage to nearline after one year, change nearline to coldline after three years, and delete items after seven years. These are good defaults, but you can create whatever rules you wish. It will not archive multiregional objects; in this case, they are assumed to be serving objects.

   ```
   {
       "lifecycle": {
           "rule": [
               {
                   "condition": {
                       "age:": 365,
                       "matchesStorageClass": ["STANDARD", "DURABLE_RECU-
   DED_AVAILBILITY"]
                   },
                   "action": {
                       "type:": "SetStorageClass",
                       "storageClass": "NEARLINE"
                   }
               },
   ```

```
                {
                    "condition": {
                        "age:": 1096,
                        "matchesStorageClass": ["NEARLINE"]
                    },
                    "action": {
                        "type:": "SetStorageClass",
                        "storageClass": "COLDLINE"
                    }
                },
                {
                    "condition": {
                        "age:": 2555,
                        "matchesStorageClass": ["COLDLINE"]
                    },
                    "action": {
                        "type:": "Delete"
                    }
                }
            ]
        }
    }
```

3. Apply the policy to the bucket:

```
gsutil lifecycle set lifecycle_config.json gs://$BUCKET_NAME
```

4. Check the lifecycle:

```
gsutil lifecycle get gs://$BUCKET_NAME
```

5. To remove lifecycle management, you apply a lifecycle config with no rules:

```
{
    "lifecycle": {
        "rule": []
    }
}
```

6. Apply:

```
gsutil lifecycle set no_lifecycle.json gs://$BUCKET_NAME
```

Discussion

Lifecycle management is a powerful, simple way to save storage costs and manage a large fleet of objects. Understanding different archival classes, in particular minimum costs for archival classes like `NEARLINE` and `COLDLINE`, is important. For example, you pay for a minimum storage duration of 90 days for `COLDLINE`. If you delete an object one minute after creating it, you will still pay as if the object had existed for 90 days. For this reason, and the fact that some storage policies can permanently delete data, tight control and review of these policies are prudent.

7.5 Creating and Restoring from Persistent Disk Snapshots in GCE

Problem

You want a reliable and easy way to back up data sitting on Google Compute Engine (GCE) VMs. You also need to be able to restore from these backups.

Solution

GCE Persistent Disk (PD) snapshots provide seamless snapshotting of your VM's data and an easy way to create new VMs from old PD snapshots in the event of a recovery scenario.

1. First, we'll create a new VM. Run the following on CLI or Cloud Shell:

   ```
   PROJECT_ID=<INSERT PROJECT>
   gcloud compute --project=$PROJECT_ID instances create pd-snapshot-test \
       --zone=us-central1-a --machine-type=e2-medium \
       --scopes=https://www.googleapis.com/auth/cloud-platform \
       --image=ubuntu-2004-focal-v20210315 --image-project=ubuntu-os-
         cloud \
       --boot-disk-size=100GB
   ```

2. SSH onto the VM. Create a file in your *$HOME* directory by running the following:

   ```
   gcloud compute ssh pd-snapshot-test --zone us-central1-a
   ```

   and then:

   ```
   echo "Hello World!" >> my_file.txt
   cat my_file.txt
   ```

3. Now from the Navigation menu, open the SNAPSHOTS section of the Compute Engine menu, shown in Figure 7-1.

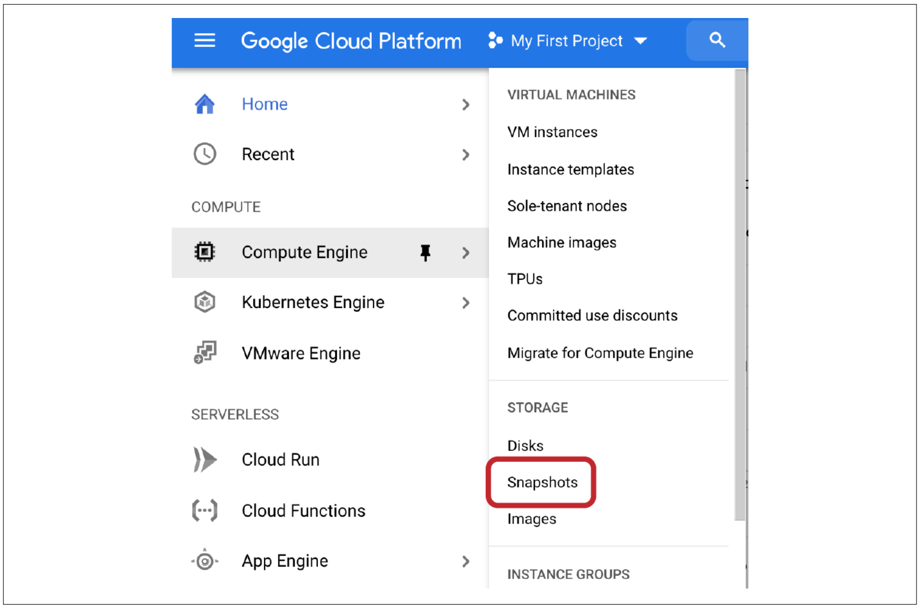
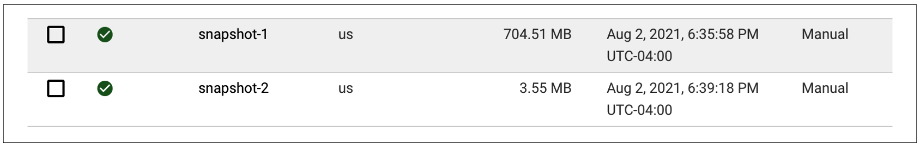


Figure 7-1. Snapshots menu under Compute Engine

4. Create a new snapshot using the source disk, which should be the same name as your VM. Leave everything else as default. Call it *snapshot-1*.
5. It will take a few minutes for the initial snapshot to take place. While you wait, take a look at the CREATE SNAPSHOT SCHEDULE tab.
6. To demonstrate an incremental snapshot, return to your VM and create a new file:

   ```
   echo "Hello Universe!" >> my_second_file.txt
   cat my_second_file.txt
   ```

7. As before, create a new snapshot and call it *snapshot-2*. You'll notice the second snapshot is much smaller than the first, in Figure 7-2, because it is incremental.



☐	✓	snapshot-1	us	704.51 MB	Aug 2, 2021, 6:35:58 PM UTC-04:00	Manual
☐	✓	snapshot-2	us	3.55 MB	Aug 2, 2021, 6:39:18 PM UTC-04:00	Manual

Figure 7-2. The two snapshots; note the second one is smaller

8. Delete *snapshot-1*, and you'll see that *snapshot-2* now is larger, as in Figure 7-3. Compute Engine automatically retains the data needed for "*snapshot-2*" without any intervention from you.

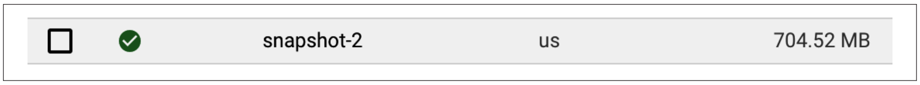


Figure 7-3. The collapsed snapshot-2

9. Delete the source VM to simulate a full outage or loss of data from the CLI. Note that we are deleting the source Persistent Disk as well.

    ```
    gcloud compute instances delete pd-snapshot-test
        --zone=us-central1-a --delete-disks=all
    ```

10. From the SNAPSHOT page, select snapshot-2 and click CREATE INSTANCE. Name your instance *pd-snapshot-restore*. If you stored your snapshot in a multiregional bucket, you can choose any zone in that multiregion. Otherwise, make sure to choose a zone in the same region as your original VM.
11. SSH back onto this new VM and test for the existence of the files.

    ```
    gcloud compute ssh pd-snapshot-restore --zone us-central1-a

    cat my_file.txt
    cat my_second_file.txt
    ```

12. Delete your instance to avoid charges.

    ```
    gcloud compute instances delete pd-snapshot-restore --zone us-central1-a
    ```

Discussion

Persistent disks are a great backup tool. They are easy to create and schedule. Also, intermediate backups save the delta rather than require the entire disk size for each backup. As you delete intermediate backups, perhaps due to a lifecycle policy, more recent backups will still be valid, as shown in this recipe. You can also configure backups to connect to the OS, in the case of Linux, and flush pending writes to disk for an application-consistent snapshot.

7.6 Using Interleaved Tables in Your Cloud Spanner Database

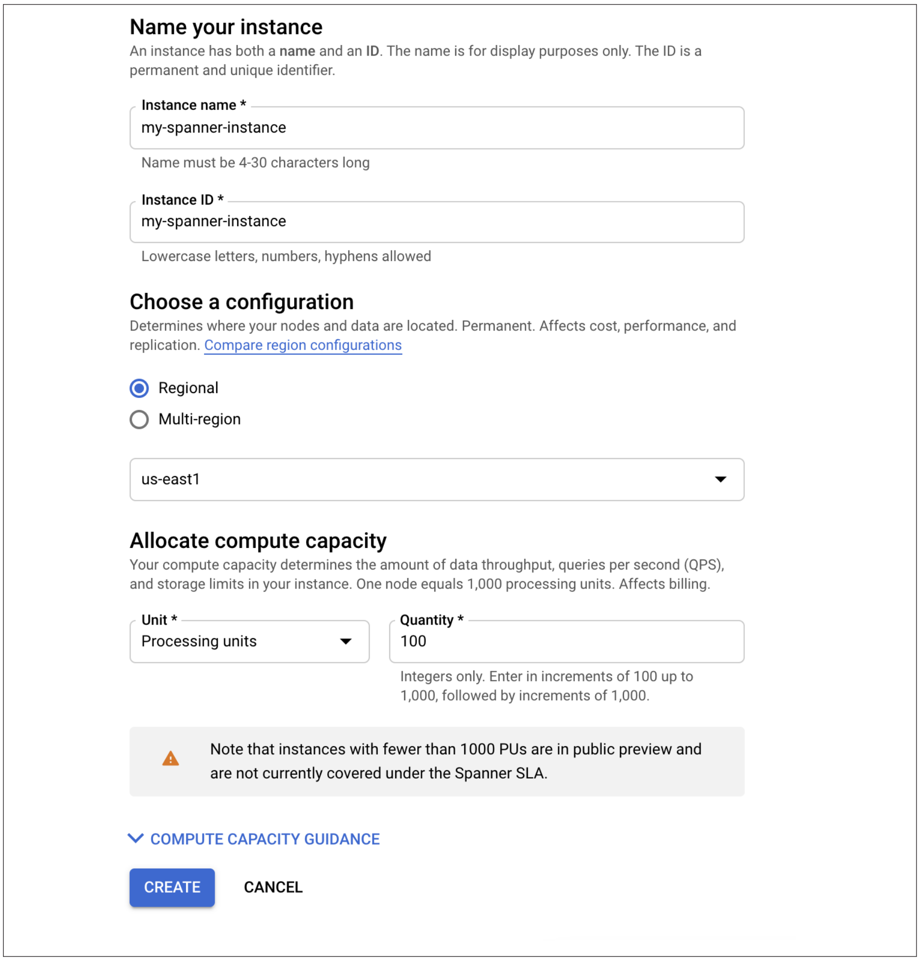
Problem

You want to speed up queries in Cloud Spanner, and you have queries that consistently reference data from multiple tables that share the same primary key.

Solution

Cloud Spanner, Google's global, distributed, serverless relational database management system (RDBMS), allows you to leverage interleaved tables to co-located data on disk from multiple tables that share the same primary key. As your data grows, it will be processed on more and more workers, so having data that is frequently queried together has speed advantages. This is an alternative to using foreign keys.

1. First, we'll create a Cloud Spanner instance and database. Navigate to the Spanner page in the Google Cloud Console and then choose CREATE INSTANCE.
2. Create an instance with the properties shown in Figure 7-4. It is important to keep processing units low to avoid unnecessary charges.


Name your instance
An instance has both a **name** and an **ID**. The name is for display purposes only. The ID is a permanent and unique identifier.

Instance name *: my-spanner-instance
Name must be 4-30 characters long

Instance ID *: my-spanner-instance
Lowercase letters, numbers, hyphens allowed

Choose a configuration
Determines where your nodes and data are located. Permanent. Affects cost, performance, and replication. Compare region configurations

[✓ Regional] Multi-region

us-east1

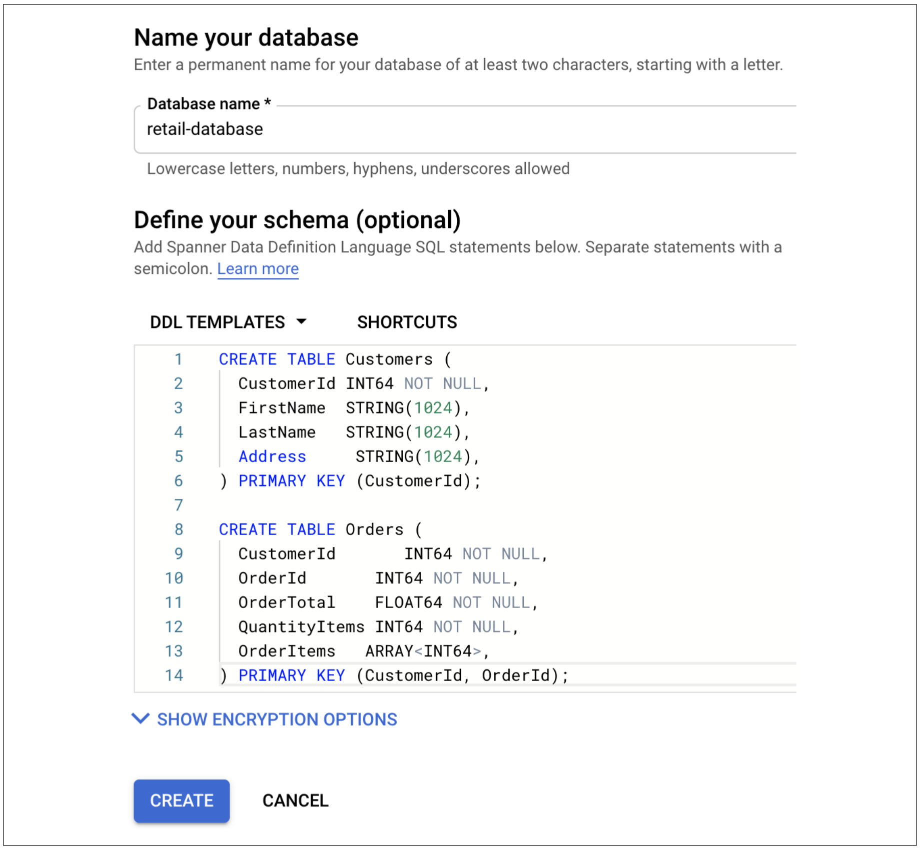
Allocate compute capacity
Your compute capacity determines the amount of data throughput, queries per second (QPS), and storage limits in your instance. One node equals 1,000 processing units. Affects billing.

Unit *: Processing units
Quantity *: 100
Integers only. Enter in increments of 100 up to 1,000, followed by increments of 1,000.

Note that instances with fewer than 1000 PUs are in public preview and are not currently covered under the Spanner SLA.

Figure 7-4. Spanner instance creation dialog box

3. Create a new database and add the following Data Definition Language (DDL) statements, as shown in Figure 7-5, to create non-interleaved tables. This uses foreign keys and a standard star schema.



Name your database

Enter a permanent name for your database of at least two characters, starting with a letter.

Database name *

retail-database

Lowercase letters, numbers, hyphens, underscores allowed

Define your schema (optional)

Add Spanner Data Definition Language SQL statements below. Separate statements with a semicolon. Learn more

```
CREATE TABLE Customers (
  CustomerId INT64 NOT NULL,
  FirstName  STRING(1024),
  LastName   STRING(1024),
  Address     STRING(1024),
) PRIMARY KEY (CustomerId);

CREATE TABLE Orders (
  CustomerId       INT64 NOT NULL,
  OrderId        INT64 NOT NULL,
  OrderTotal     FLOAT64 NOT NULL,
  QuantityItems INT64 NOT NULL,
  OrderItems    ARRAY<INT64>,
) PRIMARY KEY (CustomerId, OrderId);
```

Figure 7-5. Creating new foreign key tables

4. On the right, click the Query tool to open an interactive query editor. Run the following SQL to populate some data. We are creating two users and three total orders, with user 1 (Rui) having two orders.

```
INSERT INTO
  Customers (CustomerId,
    FirstName,
    LastName,
    Address)
VALUES
  (1, "Rui", "Costa", "123 Main Street"),
  (2, "Drew", "Hodun", "456 Park Ave"):

INSERT INTO
  Orders (OrderId,
    CustomerId,
    OrderTotal,
    QuantityItems,
    OrderItems)
  VALUES
    (1, 1, 52.34, 2, [398473, 47402]),
    (2, 1, 983.12, 3, [934773, 304983, 3872]),
    (3, 2, 10.15, 1, [3872])
```

5. Now run an aggregation query to see how much each user has spent across all their orders. My query returned 1035.46 and 10.15, respectively.

```
SELECT
  Customers.CustomerId,
  Customers.FirstName,
  Customers.LastName,
  SUM(Orders.orderTotal) AS total_spent
FROM
  Orders
JOIN
  Customers
ON
  Orders.CustomerId = Customers.CustomerId
GROUP BY
  Customers.CustomerId,
  Customers.FirstName,
  Customers.LastName
```

6. Now, this query is likely happening on a single worker node, in memory. If I had had a very large data set, such that it was being split across nodes, and large analytics queries, a lot of data would have to be passed between nodes. If we switch from foreign keys to interleaved tables, we guarantee this data is stored together and usually worked on by the same worker's process, improving query speed.
7. Click the Overview tab and then Write DDL to delete these tables. Re-create the database as before, but with this new SQL. Note the `INTERLEAVE` statement.

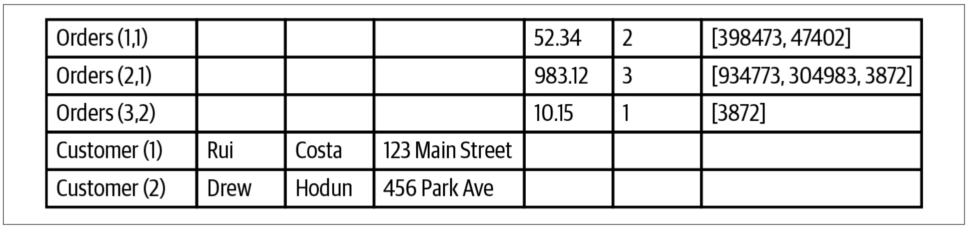
```
DROP TABLE Orders;
DROP TABLE Customers;

CREATE TABLE Customers (
  CustomerId INT64 NOT NULL,
  FirstName  STRING(1024),
  LastName   STRING(1024),
  Address     STRING(1024),
) PRIMARY KEY (CustomerId);

CREATE TABLE Orders (
  CustomerId INT64 NOT NULL,
  OrderId       INT64 NOT NULL,
  OrderTotal    FLOAT64 NOT NULL,
  QuantityItems INT64 NOT NULL,
  OrderItems   ARRAY<INT64>,
) PRIMARY KEY (CustomerId, OrderId),
  INTERLEAVE IN PARENT Customers ON DELETE CASCADE;
```

8. Re-insert the data and rerun the query as before.
9. Delete your database and instance to prevent long-term charges.

Discussion

Interleaved tables store rows physically under their associated parent row, greatly speeding up some queries in Cloud Spanner. In this case, if you wanted to perform general analytics on a given customer (for example, if you want to know what their total spend for last year is, or how many items they've ever bought), these analytic queries would complete much faster with interleaved tables because all the necessary data is physically co-located. The alternative is that the database engine would have to perform a standard key join on CustomerId to find all the orders from the order table, which might be a more expensive (slower) operation if the table is organized by timestamp, for example.

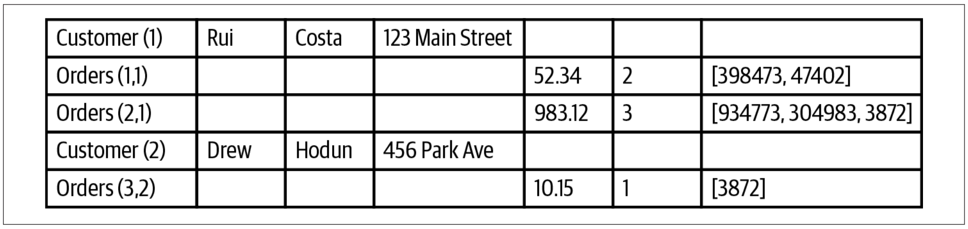
In a foreign key model, your data would be stored like the image in Figure 7-6, and potentially on different nodes:



Orders (1,1)				52.34	2	[398473, 47402]
Orders (2,1)				983.12	3	[934773, 304983, 3872]
Orders (3,2)				10.15	1	[3872]
Customer (1)	Rui	Costa	123 Main Street			
Customer (2)	Drew	Hodun	456 Park Ave			

Figure 7-6. Data as stored in foreign key tables

With interleaved tables, your data is stored as shown in Figure 7-7, improving speed through data locality.



Customer (1)	Rui	Costa	123 Main Street			
Orders (1,1)				52.34	2	[398473, 47402]
Orders (2,1)				983.12	3	[934773, 304983, 3872]
Customer (2)	Drew	Hodun	456 Park Ave			
Orders (3,2)				10.15	1	[3872]

Figure 7-7. Data as stored in interleaved tables

7.7 Locking Down Firestore Database So a User Can Edit Only Their Data

Problem

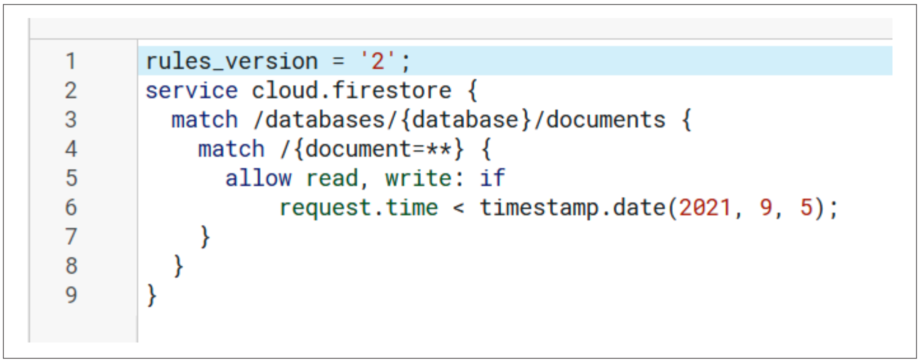
You want to secure Firestore so that an authenticated user can only create a document or edit their previous documents.

Solution

When you start working with Firestore, Google Cloud's serverless document database, security rules are open if you select Test Mode, so you'll want to lock it down early on. Firestore is often, but not always, used with Firebase, Google Cloud's platform for building and running mobile and web applications.

Firestore security rules not only can filter on whether a user is authenticated, but match their user ID to documents in the database, allowing for a simple per-user data authentication model. Also note *Firestore* is not *FILEstore, a managed, mountable file storage service.*

1. Create a Firebase project from the console to host your Firestore database. You can have one Firebase project per Google Cloud project. Go to *https://console.firebase.google.com*. Click New Project and select your Google Cloud project.
2. Choose Pay As You Go and select the defaults.
3. When the project is ready, select the Firestore tab.
4. Click Create Database and use Test Mode.
5. Open the Rules section to see the default rules that have been granted to the database, as in Figure 7-8. In this case, it is wide open for one month as you start development.



```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
          request.time < timestamp.date(2021, 9, 5);
    }
  }
}
```

Figure 7-8. Default Firestore ACL rules in test mode

6. Replace this config with the following new config. This config allows users to edit their User document (and only theirs) by matching the authenticated user ID with the user ID in the document database.

```
rules_version = '2';

service cloud.firestore {

    // Matches to the default database in the project - currently this
is the only database
    match /databases/{database}/documents {

        // Matches the userId on the authenticated request with the
userId document
        // in the users collection
        // Otherwise, allows authenticated users to create their docu-
ment
        match /users/{userId} {
            allow read, update, delete: if request.auth != null &&
request.auth.uid == userId;
            allow create: if request.auth != null;
        }
    }
}
```

7. Click PUBLISH to apply the rules. Now authenticated users are allowed to create, read, update, and delete their profile if it is stored in the /users/ collection, but nothing else.

Discussion

Firestore rules and validation provide robust but concise functionality for controlling who can edit or read what in the Firestore database. This includes setting public access for certain fields and validating that newly written input data is formatted correctly, as in the preceding example. Note that Firestore rules are not filters—that is, in the preceding case, you could not run a query as this user to return all user documents and expect the security rule to return only this user's document. This query would fail since some (basically all other) documents would not be accessible to the user. You would need to query for this user's document.

CHAPTER 8

BigQuery and Data Warehousing

With more enterprises leaning on real-time data and analytics to drive business decisions, data warehousing techniques are becoming more critical. As Google Cloud's serverless, petabyte-scale data warehouse, BigQuery is often your first and last stop for data storage, large-scale analytics, and even SQL-based machine learning models. As a serverless service, there are no clusters to create. You simply upload your data to BigQuery and start querying.

BigQuery is also very cost effective, since compute and storage are separated and can scale separately. If you never query your data, you are only charged the storage costs. But when you do run queries, you have access to a huge amount of serverless compute to process your data quickly. And you pay only for the compute used when you query, instead of paying for idle workers in a cluster.

The following recipes show examples of implementing data loading, scalable data querying, and streaming in BigQuery. Included are tips and tricks beyond standard SQL skills, some of which are specific to the BigQuery service and implementation. Several recipes will also use the `bq` command-line tool covered in Chapter 1.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

8.1 Using Cloud Console to Run a BigQuery Query

Problem

You want to get started with BigQuery quickly.

Solution

The Google Cloud Console has a full-feature SQL UI just for BigQuery, where you can browse and create data sets, run SQL queries, and schedule data transfers.

1. From the Google Cloud Console, open BigQuery, as shown in Figure 8-1.

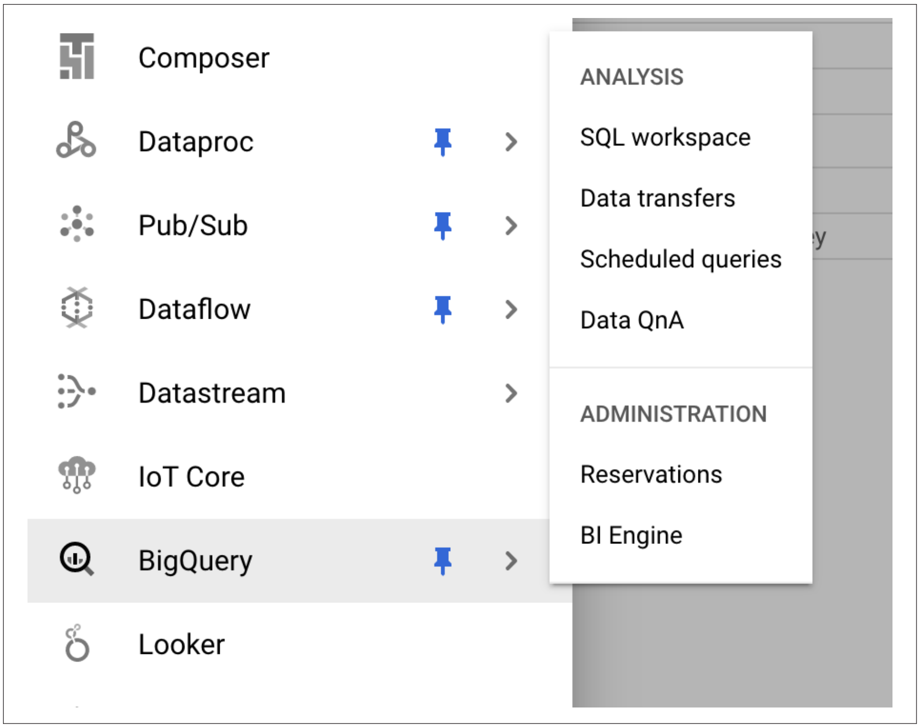
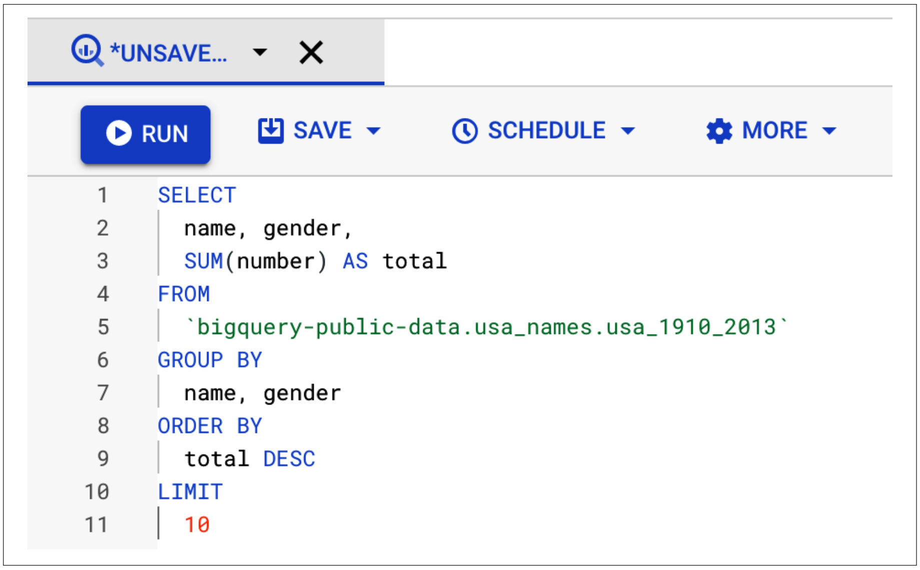


Figure 8-1. Opening the BigQuery console from the navigation menu

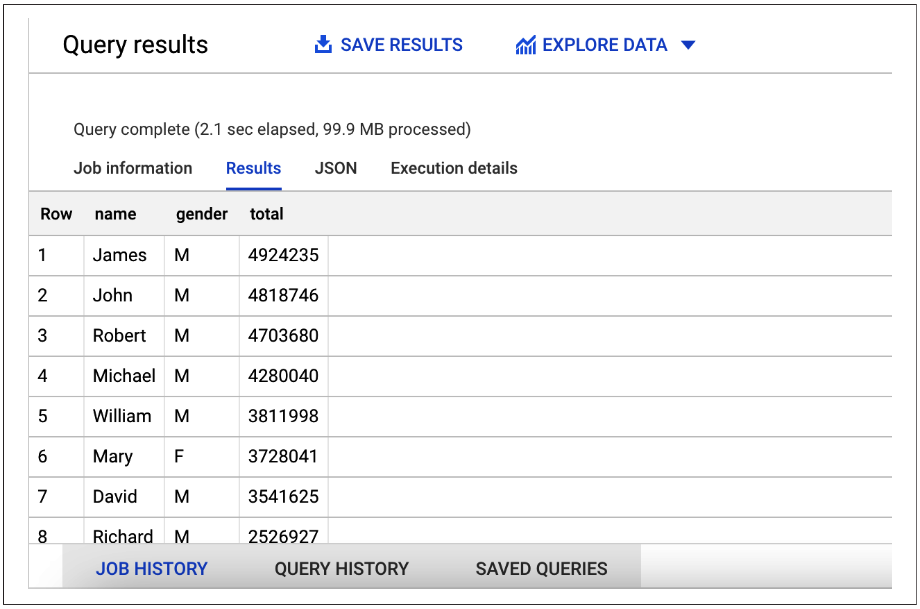
2. A query window is already open. Write this query as shown in Figure 8-2. When you're done, press Ctrl+Shift+F to auto-format the query and then click Run.



```sql
SELECT
  name, gender,
  SUM(number) AS total
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  name, gender
ORDER BY
  total DESC
LIMIT
  10
```

Figure 8-2. A simple BigQuery query in the console

3. You should see the results in a couple of seconds, as shown in Figure 8-3.



Query complete (2.1 sec elapsed, 99.9 MB processed)

Row	name	gender	total
1	James	M	4924235
2	John	M	4818746
3	Robert	M	4703680
4	Michael	M	4280040
5	William	M	3811998
6	Mary	F	3728041
7	David	M	3541625
8	Richard	M	2526927

Figure 8-3. BigQuery query results

4. Try running the query again. You will see that it happens much faster, because BigQuery caches query results for up to 24 hours. You can turn this off in Query Settings—particularly useful if you want to benchmark query performance.

Discussion

The BigQuery console offers a great place to start using the service. You can explore data, create data sets, browse the project's query history, and even save queries to share with your teammates. A few helpful hotkeys are listed in Table 8-1.

Table 8-1. BigQuery hotkeys

Key Press Combo	Action
Ctrl+Shift+F	Auto-formats the query
Ctrl+Enter	Executes the entire query window
Ctrl+E (on a selection)	Execute only the select SQL

BigQuery caching can be a great help at speeding up data exploration and avoiding costs for queries you keep repeating. You should not rely on caching in a production setting, however, because these results are not guaranteed to persist for 24 hours. If you really have a need for fast, cached results, materialize your data in a table that you periodically refresh. In addition, the caching mechanism is rather simplistic: it hashes the text of the full query, so even adding a space will cause a cache miss. Subqueries, views, or any portion of the query are also not cached; just the query as a whole is cached.

8.2 Loading Data to BigQuery from CSV

Problem

You have some local comma-separated values (CSV) data you'd like to load to BigQuery for analysis.

Solution

BigQuery has many ways to load data easily. This example will use the UI.

1. Download a sample file. Run the following `gsutil` command to download a sample CSV from Google Cloud Storage to your workstation. You will need to have gcloud installed.

```
gsutil cp gs://cloud-samples-data/bigquery/us-states/us-states-by-
date.csv.
```

2. As before, open the BigQuery console.

3. Create a new data set. Click the three dots near your project name on the left of the browser and select Create Dataset, as shown in Figure 8-4.

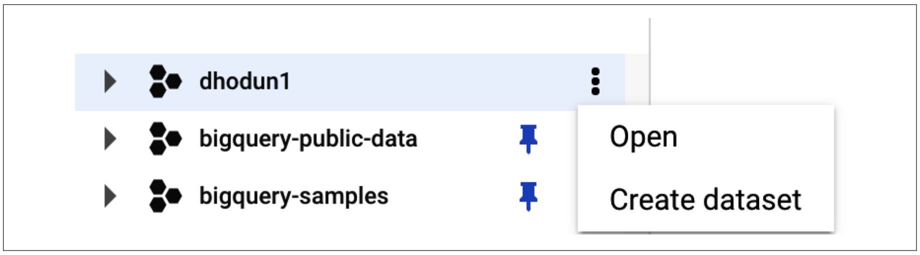
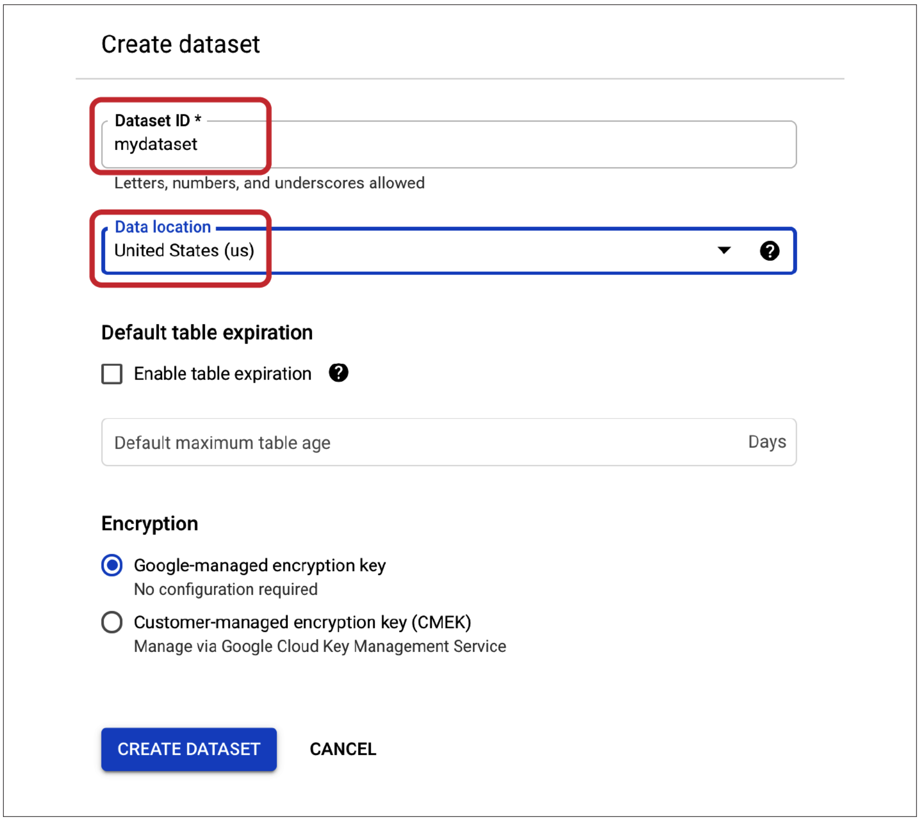


Figure 8-4. Creating a data set in the data set browser

4. Name your data set **mydataset** and choose US as the processing location, as in Figure 8-5. This cannot be changed afterward.



Create dataset

Dataset ID *
mydataset
Letters, numbers, and underscores allowed

Data location
United States (us)

Default table expiration

☐ Enable table expiration

Default maximum table age Days

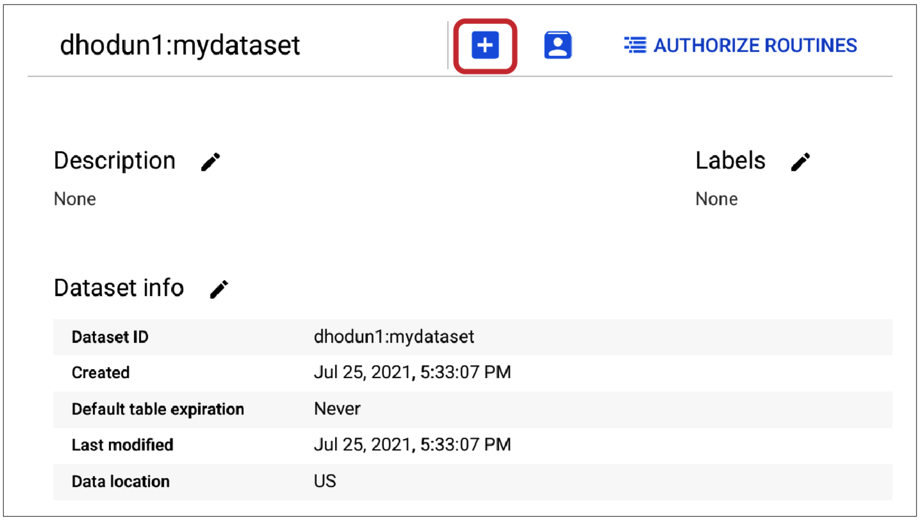
Encryption

● Google-managed encryption key
No configuration required

○ Customer-managed encryption key (CMEK)
Manage via Google Cloud Key Management Service

Figure 8-5. Creating a new data set

5. You will now see the data set under your project in the browser. Click the three dots on the data set and open it. Click Create Table, as shown in Figure 8-6.



Description: None

Labels: None

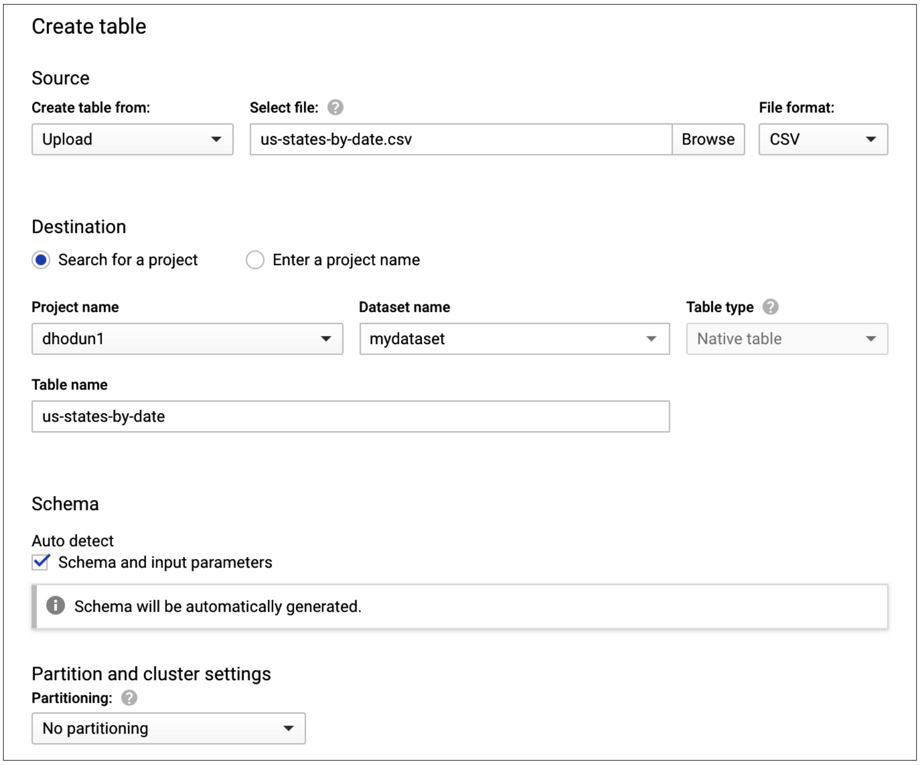
Dataset ID	dhodun1:mydataset
Created	Jul 25, 2021, 5:33:07 PM
Default table expiration	Never
Last modified	Jul 25, 2021, 5:33:07 PM
Data location	US

Figure 8-6. Creating a new table

6. Enter the following data in the dialog box.

Option	Value
Create table from:	Upload
Select file:	The file you previously downloaded
File format:	CSV
Dataset name:	mydataset
Table name:	us-states-by-date
Schema Auto-detect:	Check Yes

The results should look something like Figure 8-7. Click Create Table.


Create table

Source

Create table from: Upload

Select file: us-states-by-date.csv

Browse

File format: CSV

Destination

[✓ Search for a project] Enter a project name

Project name: dhodun1

Dataset name: mydataset

Table type: Native table

Table name: us-states-by-date

Schema

Auto detect

[✓ Schema and input parameters]

Schema will be automatically generated.

Partition and cluster settings

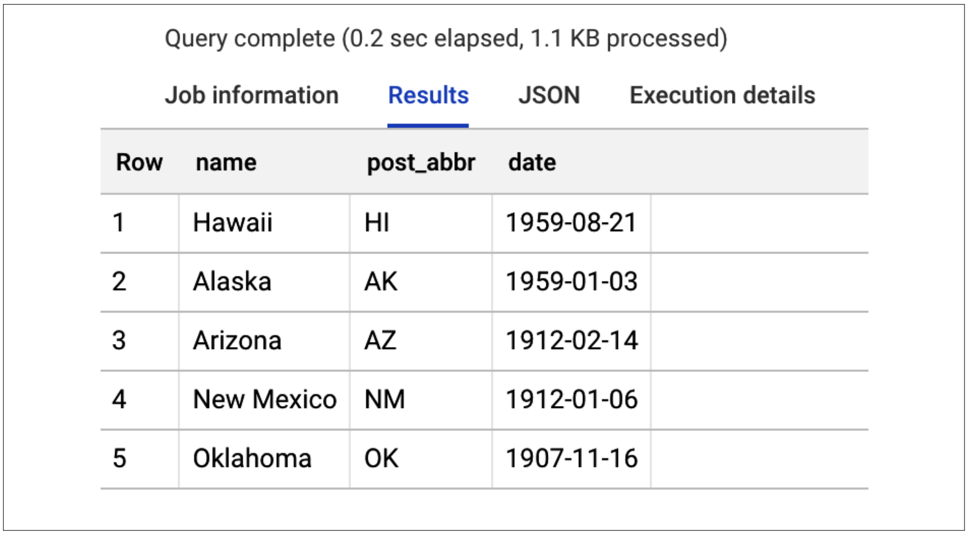
Partitioning: No partitioning

Figure 8-7. Create Table dialog box with options filled in

7. It should take only a few moments for the table to be created. When it is, you can open the table, inspect the schema that was generated, and even view some of the data with the Preview tab. Last, click the Query button to query your data. Run the following SQL to find the five newest states in the United States, using your project name instead:

```
SELECT *
FROM
  `dhodun1.mydataset.us-states-by-date`
ORDER BY
  date DESC
LIMIT
  5
```

Your results should look like Figure 8-8.


Query complete (0.2 sec elapsed, 1.1 KB processed)

Row	name	post_abbr	date
1	Hawaii	HI	1959-08-21
2	Alaska	AK	1959-01-03
3	Arizona	AZ	1912-02-14
4	New Mexico	NM	1912-01-06
5	Oklahoma	OK	1907-11-16

Figure 8-8. The five newest states in the United States

Discussion

This demonstrated one of the simplest upload use cases. You can upload using many data types, including CSV, AVRO, JSON, and Parquet. Often, you will be uploading from GCS or S3 (the comparable service on AWS) rather than from your desktop. You can also set up BigQuery data transfer jobs to pull in data regularly from external sources on a scheduled basis. You can programmatically write to BigQuery as a sink from many data processing frameworks, such as Spark, Dataflow, Data Fusion, and other BigQuery queries.

8.3 Building a Pivot Table in BigQuery

Problem

You want to build a pivot table in BigQuery using SQL.

Solution

BigQuery now supports PIVOT as an operator natively in SQL. Rather than aggregate results in one long table output, sometimes it is helpful to pivot the data, that is, calculate aggregates across distinct categories and give each category its own column.

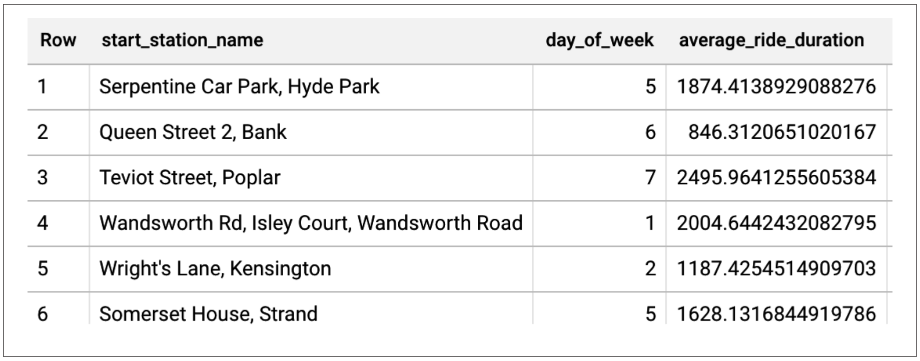
1. Open the BigQuery UI and run the following query. This query calculates the average ride length for each start_station for each day of the week; "1" represents Sunday, "2" for Monday, and so on. For this and many of the following recipes,

we will be using a public table containing ride data from the London bicycle share program.

> You may need to change the processing location to "EU" by clicking Query Settings and updating Processing Location.

```
SELECT
  start_station_name,
  EXTRACT(DAYOFWEEK
  FROM
    end_date) AS day_of_week,
  AVG(duration) AS average_ride_duration
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
GROUP BY
  start_station_name,
  day_of_week
```

You'll notice we have some 6,000 rows and the output isn't particularly readable, as shown in Figure 8-9. What we really want to see is the data displayed for each start_station and day_of_week pair, where the days of the week are columns.



Row	start_station_name	day_of_week	average_ride_duration
1	Serpentine Car Park, Hyde Park	5	1874.4138929088276
2	Queen Street 2, Bank	6	846.3120651020167
3	Teviot Street, Poplar	7	2495.9641255605384
4	Wandsworth Rd, Isley Court, Wandsworth Road	1	2004.6442432082795
5	Wright's Lane, Kensington	2	1187.4254514909703
6	Somerset House, Strand	5	1628.1316844919786

Figure 8-9. Unpivoted output

2. You can accomplish this with a `PIVOT` clause. The `PIVOT` clause first needs a raw data set without any aggregates applied. Then it requires the aggregate function and target column, followed by the categories to pivot on. Run the following query to pivot the data and arrange the averages so that each day_of_week has its own column:

```
SELECT *
FROM (
  SELECT
```

```
    start_station_name,
    EXTRACT(DAYOFWEEK FROM end_date) AS day_of_week,
    duration
  FROM
    `bigquery-public-data.london_bicycles.cycle_hire`
)
PIVOT
(
  AVG(duration) AS average_ride_duration
  FOR day_of_week IN (1,2,3,4,5,6,7)
)
```

You should see that for most stations, the weekend rides (Days 1 and 7) have a longer duration.

Discussion

Often, pivoting is a display function supported in business intelligence (BI) tools, but sometimes you may want to format your data this way, particularly if you are doing rapid data exploration natively in BigQuery. This previously was trickier to accomplish, requiring CASE statements or stored procedures. An UNPIVOT operator now also exists, which does the opposite: it turns columns back into rows.

8.4 Adding Partitioned and Clustered Columns to an Existing Table

Problem

You have a BigQuery table in which you need to add a partitioned column and clustered column to increase query performance and decrease query costs.

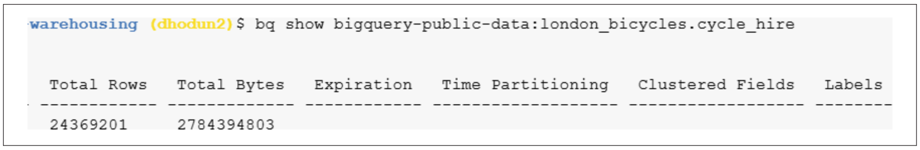
Solution

BigQuery SQL allows you to create a new table with the partitioned and clustered columns and populate it in the same statement with an AS clause. Note that it will be stored in an entirely new table.

1. Validate that the source table doesn't have partitioned or clustered columns. Run the following in your local terminal with gcloud installed or in Cloud Shell:

   ```
   bq show bigquery-public-data:london_bicycles.cycle_hire
   ```

   Notice the lack of columns on the right in Figure 8-10.



```
warehousing (dhodun2)$ bq show bigquery-public-data:london_bicycles.cycle_hire


  Total Rows   Total Bytes   Expiration   Time Partitioning   Clustered Fields   Labels
 ------------ ------------- ------------ ------------------- ------------------ --------
  24369201     2784394803
```

Figure 8-10. Public cycle_hire table without partitioned or clustered columns. Notice the right side of the output

2. Dry-run a time-based query to see how much data is scanned. Execute this code on the command line or Cloud Shell. You can also copy and paste it from the repo code. Notice the `--dry-run flag`, which is used to report on whether the query is parsed properly, whether you have access to the underlying data sets, and how much data will be scanned.

```
QUERY='
SELECT
  duration,
  bike_id,
  start_date,
  start_station_name,
  end_date,
  end_station_name
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
WHERE
  EXTRACT(DATE FROM start_date ) = "2016-04-03"
  AND EXTRACT(DATE FROM end_date ) = "2016-04-03"
ORDER BY
  duration DESC
LIMIT
  5
'

bq query \
--use_legacy_sql=false \
--location=EU \
--dry_run \
$QUERY
```

The output should look something like this and validate that the query will process the entire 2.2 GB in the table:

```
Query successfully validated. Assuming the tables are not modified, run-
ning this query will process 2197696867 bytes of data.
```

3. If you haven't already created a data set in the EU region, create it now from the command line.

```
bq mk --location=eu mydataset_eu
```

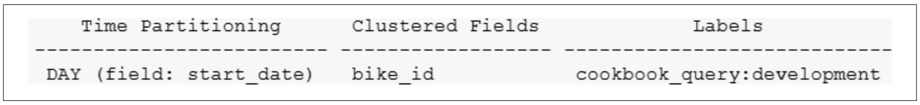
4. Now, in the BigQuery UI, create a new table in your data set, indicating the partitioned and clustered columns. Run this SQL:

```
CREATE OR REPLACE TABLE
  mydataset_eu.cycle_hire_partitioned_clustered
PARTITION BY
  DATE(start_date)
CLUSTER BY
  bike_id
OPTIONS( expiration_timestamp=TIMESTAMP "2025-01-01 00:00:00 UTC",
     description="My partitioned and clustered cycle_hire table",
     labels=[("cookbook_query", "development")] )
AS
SELECT
  *
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
```

You may need to change the processing location to EU by clicking Query Settings and updating Processing Location.

5. Validate that the destination table does have partitioned or clustered columns. Run the following from the CLI, and you'll see the schema output, as in Figure 8-11:

```
bq show mydataset_eu.cycle_hire_partitioned_clustered
```



```
    Time Partitioning       Clustered Fields              Labels
 ------------------------- ------------------ ----------------------------
  DAY (field: start_date)   bike_id            cookbook_query:development
```

Figure 8-11. Your new cycle_hire, now with partitioned or clustered columns. See the right side of the output

6. Dry-run the query on the new table to see how much will be scanned:

```
QUERY='
SELECT
  duration,
  bike_id,
  start_date,
  start_station_name,
  end_date,
  end_station_name
FROM
  `mydataset_eu.cycle_hire_partitioned_clustered`
WHERE
  EXTRACT(DATE FROM start_date ) = "2016-04-03"
  AND EXTRACT(DATE FROM end_date ) = "2016-04-03"
ORDER BY
  duration DESC
LIMIT
  5
'

bq query \
--use_legacy_sql=false \
--location=EU \
--dry_run \
$QUERY
```

The output should look something like this and validate that the query will process only 2 MB (you can remove `--dry-run` to run the query, if you like):

```
Query successfully validated. Assuming the tables are not modified, run-
ning this query will process upper bound of 2285452 bytes of data.
```

7. Don't forget to delete your table and data set to avoid charges.

Discussion

Although BigQuery is a serverless data warehouse with few knobs for tuning outside of how you author your SQL, partitioning and clustering are two options available for increasing performance and decreasing costs.

Partitioning splits your table into smaller segments from an internal BigQuery standpoint, most commonly based on date or time. When you run a query with a filter clause, such as `WHERE EXTRACT(DATE FROM start_date ) = "2016-04-03"`, BigQuery knows not to read in the data from partitions that don't match the filter, greatly improving performance and decreasing query cost, since this cost is based on the data actually scanned. You can partition on only one column, and it needs to be a date, time, or integer.

Specifying clustered columns allows BigQuery to organize and order table data automatically, based on these columns. Related data is co-located, resulting in faster data. The order that you specify for clustered columns is important, because this order is used to decide which column to sort first. Similar to partitioning, clustering can improve query performance when you are filtering or aggregating on these columns.

This example also showed use of the dry-run capability of the `bq` command-line tool to determine the query cost before running it. This is also available in the BigQuery UI.

8.5 Adding Clustering to a Table That Can't or Shouldn't Be Partitioned

Problem

You have a BigQuery table to which you'd like to add clustered columns. Clustering requires the table to be partitioned, however, and your table doesn't have a DATE-TIME or INTEGER column that is an obvious partitioning candidate. In addition, if your table is smaller and you are adding less than 10 GB per day, adding partitioned columns may not be desirable.

Solution

You can partition on the hidden ingestion time column, which is preferred if you are ingesting more than 10 GB a day. If you have smaller amounts of data, create an optional fake date column to partition on and just leave it as NULL. Both of these then allow you to cluster fields as normal. Here we show the hidden column method.

We will be using the same data set as in the previous recipe, the London cycle_hire data set. Although it does have a couple of obvious timestamp columns to partition on, the data set is small enough at 2 GB to benefit from the fake column partitioning method.

1. Run the following SQL and note the new fake column:

```
CREATE OR REPLACE TABLE
  mydataset_eu.cycle_hire_fake_partitioned_clustered (
    rental_id INTEGER,
    duration INTEGER,
    bike_id INTEGER,
    end_date TIMESTAMP,
    end_station_id INTEGER,
    end_station_name STRING,
    start_date TIMESTAMP,
    start_station_id INTEGER,
    start_station_name STRING,
    end_station_logical_terminal INTEGER,
    start_station_logical_terminal INTEGER,
    end_station_priority_id INTEGER,
    fake_date DATE )
PARTITION BY
  fake_date
CLUSTER BY
  bike_id
OPTIONS( expiration_timestamp=TIMESTAMP "2025-01-01 00:00:00 UTC",
    description="My partitioned and clustered cycle_hire table",
    labels=[("cookbook_query",
        "development")] ) AS
SELECT
  *, NULL
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
```

2. Run the following query to see the maximum duration for bike number 153:

   ```
   SELECT MAX(duration) as max_duration FROM
   `mydataset_eu.cycle_hire_fake_partitioned_clustered`
   WHERE bike_id =  153
   ```

   The query should process in the low number of megabytes, in my case, 5.3 MB.

3. Now run the same query on the original data set:

   ```
   SELECT MAX(duration) as max_duration FROM
   `bigquery-public-data.london_bicycles.cycle_hire`
   WHERE bike_id =  153
   ```

   The query will process much more data, in my case, 372 MB.

4. If you still have the table from the previous recipe around, query that one as well:

   ```
   SELECT MAX(duration) as max_duration FROM
   `mydataset_eu.cycle_hire_partitioned_clustered`
   WHERE bike_id =  153
   ```

The query will also process much more data, in my case, 360 MB.

Discussion

For a while, partitioning was the only way to speed up your BigQuery queries, aside from rewriting SQL, if that were even an option. Under the hood, partitioning actually creates mini-tables that BigQuery simply extracts for you. Since our table is only 2.2 GB, this really doesn't improve performance; in fact, it may decrease it, since BigQuery has to scan hundreds of files, one for each day. Using the fake partition column followed by clustering gives us much better performance on this smaller table. It also is an option to enable clustering on tables that, unlike this one, don't have an obvious partition column.

For larger tables that don't have an obvious partition column or tables where you are adding more than 10 GB a day, partitioning based on the insert-time pseudocolumns, `_PARTITIONDATE` or `_PARTITIONTIME`, is recommended.

8.6 Selecting the Top-1 Result

Problem

You want to return the top item from a sorted list in a BigQuery query, but your data set is particularly large and the query is slow.

Solution

You can start by using the standard `ROW_NUMBER() OVER()` windowing function and then filtering based on the first item. However, if your data set is particularly large and performing slowly because you're forcing a full sort of the data set, you can apply a BigQuery-specific trick. Using `ARRAY_AGG(x LIMIT 1)[OFFSET(0)]` will allow BigQuery to drop all data that isn't the number 1 row, increasing query performance.

1. In the BigQuery UI, run a query using the `ROW_NUMBER OVER()` windowing function. This query should succeed but will take a while.

```
SELECT
  rental_id,
  duration,
  bike_id,
  end_date
FROM (
  SELECT
    rental_id,
    duration,
    bike_id,
    end_date,
    ROW_NUMBER() OVER (ORDER BY end_date ASC) rental_num

  FROM
    `bigquery-public-data`.london_bicycles.cycle_hire )
WHERE
  rental_num = 1
```

2. Run a query using `ARRAY_AGG(x LIMIT 1)[OFFSET(0)]` instead. This query should succeed more quickly.

```
SELECT
  rental.*
FROM (
  SELECT
    ARRAY_AGG( rentals
    ORDER BY rentals.end_date ASC LIMIT 1)[OFFSET(0)] rental
  FROM (
    SELECT
      rental_id,
      duration,
      bike_id,
      end_date
    FROM
      `bigquery-public-data`.london_bicycles.cycle_hire) rentals )
```

Discussion

Although `ROW_NUMBER()` has a scalable implementation that can perform distributed sorts across multiple BigQuery workers, the entire sort needs to be calculated. In particularly large queries, using `ARRAY_AGG` allows BigQuery to drop unneeded data. This is a BigQuery-specific trick to speed up sorts when you know you need only the top-n results.

8.7 Merging Tables in BigQuery Without Duplicates

Problem

You are loading data into an existing BigQuery table that might introduce duplicate rows, and you want to deduplicate your data. This is common with repeated batch uploads or when consolidating from a streaming ingest table that might contain newer versions of a given record or aggregation.

Solution

Using the `MERGE ON...WHEN MATCHED` clause in your `CREATE TABLE` statement, you can easily indicate to BigQuery a row key on which to deduplicate and insert only the new rows into your final table.

1. Create a production table. For our sample, we will copy the public London cycle-hire data set and use the rental_id row to deduplicate.

    ```
    bq cp -f bigquery-public-data:london_bicycles.cycle_hire mydata-
    set_eu.cycle_hire
    ```

2. Create a loading table with new and duplicate data. This represents data generated by some production process that we want in our data warehouse but that will sometimes generate duplicates.

    ```
    CREATE OR REPLACE TABLE
      mydataset_eu.temp_loading_table AS (

      --Grab 5 duplicate rows
      SELECT
        *
      FROM
        mydataset_eu.cycle_hire
      LIMIT
        5)
    UNION ALL (

      --Add a new unique row
      SELECT
        111147469109,
    ```

```
        3180,
        7054,
        '2015-09-03 12:45:00 UTC',
        111,
        'Park Lane, Hyde Park',
        '2015-09-03 11:52:00 UTC',
        300,
        'Serpentine Car Park, Hyde Park',
        NULL,
        NULL,
        NULL)
```

3. Verify and note the counts in both tables. I got 24369201 and then 6 when I ran it.

    ```
    --Number of Rows in base table
    SELECT COUNT(*) FROM mydataset_eu.cycle_hire;

    --Number of Rows in loading table
    SELECT COUNT(*) FROM mydataset_eu.temp_loading_table;
    ```

4. Merge the data into your production table with an `ON` clause.

    ```
    MERGE
      mydataset_eu.cycle_hire rentals
    USING
      mydataset_eu.temp_loading_table temp
    ON
      temp.rental_id = rentals.rental_id
      WHEN NOT MATCHED
      THEN
        INSERT ROW
    ```

    The query results should say that only one row was affected.

5. Verify that only one row has been inserted. This number should be one greater than before.

    ```
    --Number of rows now in base table
    SELECT COUNT(*) FROM mydataset_eu.cycle_hire;
    ```

Discussion

This recipe shows how to insert data in a way that doesn't cause duplicates based on the specified row key. For an example of how to filter out duplicates inline in a query (but without any need for sorting in the `ARRAY_AGG` function), see the following recipe.

It's also worth noting that you can deduplicate with `SELECT DISTINCT * FROM mydataset.mytable`, which will ensure that every record is entirely unique (not just the key), but this is often a slower query.

8.8 Deduplicating Rows in BigQuery with Timestamps

Problem

Your application is inserting updated rows or aggregations into a BigQuery table rather than updating existing ones for performance reasons, and you need to select just the newest data for each key. This commonly happens in streaming applications and frameworks, such as Cloud Dataflow.

Solution

You could do this with the `ROW_NUMBER() OVER()` windowing function, sorting only the first row for each key based on `TIMESTAMP`. Or, for a more scalable approach, you could implement our `ARRAY_AGG(x LIMIT 1)[OFFSET(0)]` trick from the previous recipe so that older data is simply dropped by BigQuery workers in-flight. You will try both examples here.

1. Create a table to query. You may still have this table from the previous recipe:

```
bq --location=eu cp -f bigquery-public-data:london_bicycles.cycle_hire
mydataset_eu.cycle_hire
```

2. Examine the initial record for a particular rental ID. In this case, the record was an estimated journey, which will be updated with newer, more complete data at a later time:

```
SELECT
  *
FROM
  mydataset_eu.cycle_hire
WHERE
  rental_id = 47469109
ORDER BY
  end_date DESC
```

3. Insert a couple of new rows with the same `ride_id` but with updated duration. The `ride_id` for these records is the same as an existing entry, but the `end_date` timestamps are newer, representing newer versions of the data:

```
INSERT INTO
 mydataset_eu.cycle_hire
VALUES
  ( 47469109, 3300, 7054, '2015-09-03 12:47:00 UTC', 111, 'Park Lane,
Hyde Park', '2015-09-03 11:52:00
UTC', 300, 'Serpentine Car Park, Hyde Park', NULL, NULL, NULL ),
  ( 47469109, 3660, 7054, '2015-09-03 12:53:00 UTC', 111, 'Park Lane,
Hyde Park', '2015-09-03 11:52:00
UTC', 300, 'Serpentine Car Park, Hyde Park', NULL, NULL, NULL )
```

4. View all three records. You can see that we have three versions of this ride:

```
SELECT
  *
FROM
  mydataset_eu.cycle_hire
WHERE
  rental_id = 47469109
ORDER BY
  end_date DESC
```

5. Use `ROW_NUMBER() OVER()` to get the latest record of this `rental_id`. Notice that the duration is longer than the original entry:

```
SELECT
  * EXCEPT(row_num)
FROM (
  SELECT
    *, ROW_NUMBER() OVER (PARTITION BY rental_id ORDER BY end_date
DESC) AS row_num
  FROM
    mydataset_eu.cycle_hire )
WHERE
  row_num = 1
  AND rental_id = 47469109
```

6. Use `ARRAY_AGG` to get the latest one:

```
SELECT
  latest_record.*
FROM (
  SELECT
    rental_id,
    ARRAY_AGG(rentals ORDER BY end_date DESC LIMIT 1)[OFFSET(0)] lat-
est_record
  FROM
    mydataset_eu.cycle_hire rentals
  WHERE
    rental_id = 47469109
  GROUP BY
    rental_id )
```

7. Last, remove the `WHERE` clause and run this on the entire data set. At the moment, we have introduced duplicates only for this `rental_id`, but if you had a streaming system constantly sending updated records, this would show only the latest.

```
SELECT
  latest_record.*
FROM (
  SELECT
    rental_id,
    ARRAY_AGG(rentals ORDER BY end_date DESC LIMIT 1)[OFFSET(0)] lat-
est_record
  FROM
```

```
    mydataset_eu.cycle_hire rentals
  GROUP BY
    rental_id )
```

Discussion

This trick adds an outer SELECT clause, mostly to clean up the nested output from ARRAY_AGG, but it is a scalable implementation of cleaning up data when there are newer records. This extra SELECT clause can be hidden from the user by placing the query in View or Materialized View for end-user analysts or downstream programs to consume from.

This use case is particularly common with streaming systems or any high-throughput systems that have newer versions of data or aggregations. Although BigQuery does support ACID-compliant UPDATE and DELETE DML statements so that individual records can be mutated and updated transactionally, this is generally a costly operation, since the files backing BigQuery are immutable (an update triggers entire file rewrites), and the system overall is tuned for online analytical processing (OLAP) queries. For this reason, some scalable streaming systems opt to generate new records in BigQuery and let BigQuery perform the deduplication as noted previously.

8.9 Undeleting a Table in BigQuery

Problem

You have accidentally deleted a table in BigQuery and need to recover it.

Solution

You can use snapshot decorators with the bq tool to recover the data.

1. Create a dummy table:

```
CREATE OR REPLACE TABLE
  mydataset_eu.cycle_hire AS
SELECT
  *
FROM
  `bigquery-public-data`.london_bicycles.cycle_hire
```

2. Note the UTC time on your CLI. This command shows the number of seconds since the UNIX epoch, or 00:00:00 UTC on January 1, 1970. If this command doesn't work, you can also check out *https://www.epochconverter.com* to grab the current time.

```
date -u +%s
```

You should see something like this; take note of this time:

```
1627238475
```

3. Accidentally delete the table. Note that this is happening *after* the timestamp we grabbed.

```
DROP TABLE mydataset_eu.cycle_hire
```

4. Restore the data into a temporary staging table. Note that we multiply the UNIX seconds by 1,000 since we need milliseconds. Update the time to the one you just output, adding three zeros to the end, and then run this on the CLI.

```
bq --location=eu cp mydataset_eu.cycle_hire@1609277559000 mydata-
set_eu.cycle_hire_restored
```

5. Verify that the data is back:

```
bq head -n 5 mydataset_eu.cycle_hire_restored
```

Discussion

This fix should also work if you have already replaced the table, which occasionally happens if you have automated pipelines configured to create the table if it doesn't exist. In this situation, you can also use BigQuery time travel with `FOR SYSTEM TIME AS OF` in SQL to recover the data:

```
SELECT
 *
FROM
 mydataset.cycle_hire_restored
FOR SYSTEM TIME AS OF '2020-12-29 21:44:09.413928 UTC'
```

8.10 Streaming JSON or Avro Data into BigQuery with a Dataflow Template

Problem

You want to ingest real-time streaming data into a BigQuery table from Pub/Sub for analysis and reporting. For example, this could be data generated by real-time web application user data or Internet of Things (IOT) devices.

Solution

Configuring and executing a Cloud Dataflow template can quickly spin up a persistent, streaming dataflow job with little to no code that will read the Pub/Sub messages and write them in real time to BigQuery. Your Pub/Sub messages will need to be JSON or Avro.

1. Sample a message from the sample public taxi data topic:

   ```
   gcloud pubsub subscriptions create taxi-test-sub --topic projects/
   pubsub-public-data/topics/taxirides-realtime

   gcloud pubsub subscriptions pull projects/<your-project-id>/subscrip-
   tions/taxi-test-sub
   ```

   which will result in JSON message data:

   ```
   {"ride_id":"7128ff90-62f7-42de-a93f-67d1f9e7c713","point_idx":2401 "lat-
   itude":40.74163,"longitude":-73.95092000000001,"timestamp":
   "2021-01-06T20:56:11.93239-05:00","meter_reading":
   55.330975,"meter_increment":0.02304497,"
   ride_status":"enroute","passenger_count":2}
   ```

2. Create a destination in BigQuery tables, both for well-formed data and for a dead-letter table. A schema is needed on the BigQuery table, matching the JSON data on the Pub/Sub topic.

   ```
   cat <<EOF > schema.json
   [
         {
           "name": "ride_id",
           "type": "STRING"
         },
         {
           "name": "point_idx",
           "type": "INTEGER"
         },
         {
           "name": "latitude",
           "type": "FLOAT"
         },
         {
           "name": "longitude",
           "type": "FLOAT"
         },
         {
           "name": "timestamp",
           "type": "TIMESTAMP"
         },
         {
           "name": "meter_reading",
           "type": "FLOAT"
         },
         {
           "name": "meter_increment",
           "type": "FLOAT"
         },
         {
           "name": "ride_status",
   ```

```
            "type": "STRING"
          },
          {
            "name": "passenger_count",
            "type": "INTEGER"
          }
        ]
    EOF
    bq mk mydataset.taxi_data schema.json
    bq mk mydataset.taxi_deadletter
```

3. Create a temporary GCS bucket for the dataflow job:

```
BUCKET=gs://my-bucket
gsutil mb $BUCKET
```

4. In the Google Cloud Console, open the Dataflow page and click Create Job From Template, as in Figure 8-12.

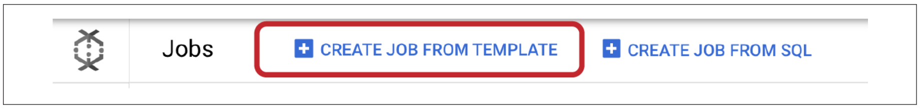
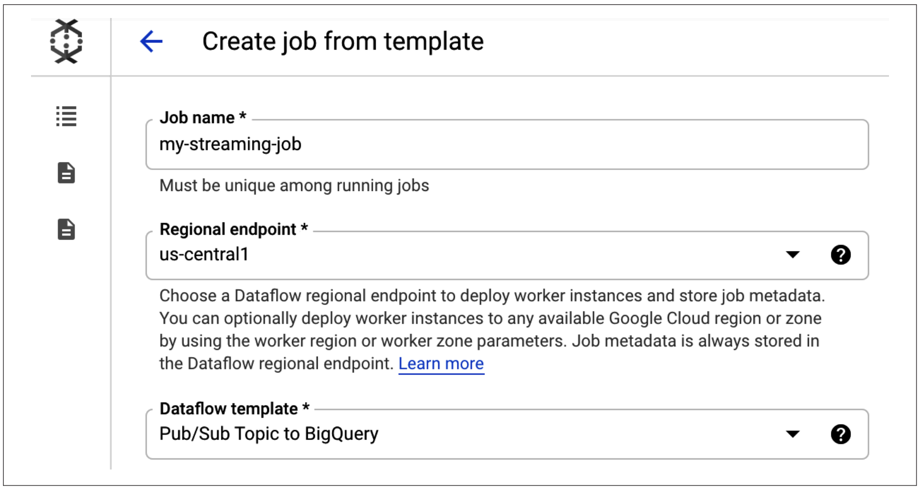


Figure 8-12. Creating a new template job in Dataflow

5. Add a job name and choose Pub/Sub Topic to BigQuery. Note the other templates that are available, as shown in Figure 8-13.



Create job from template

Job name *
my-streaming-job
Must be unique among running jobs

Regional endpoint *
us-central1
Choose a Dataflow regional endpoint to deploy worker instances and store job metadata. You can optionally deploy worker instances to any available Google Cloud region or zone by using the worker region or worker zone parameters. Job metadata is always stored in the Dataflow regional endpoint. Learn more

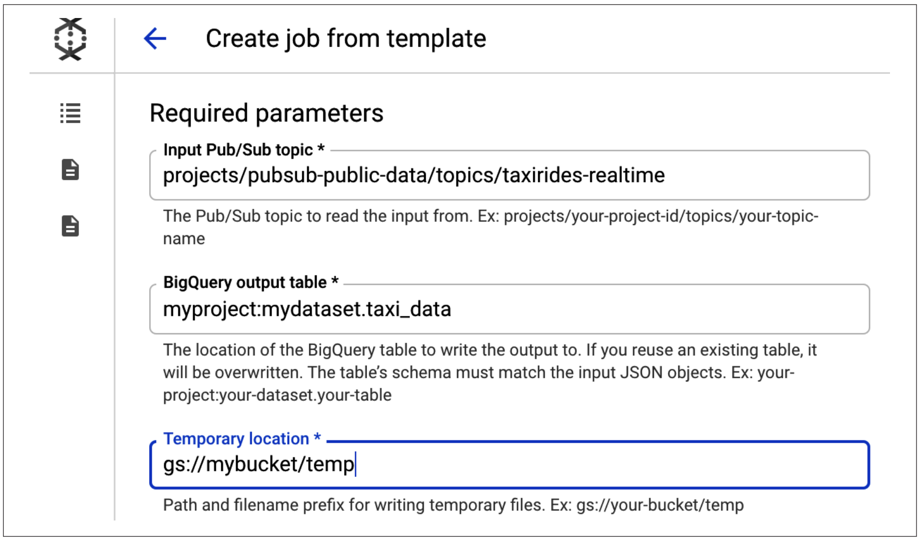
Dataflow template *
Pub/Sub Topic to BigQuery

Figure 8-13. The Dataflow template job form

6. For Input Pub/Sub Topic, point to the topic:

```
projects/pubsub-public-data/topics/taxirides-realtime
```

7. Add the BigQuery table and the temporary GCS bucket for the dataflow job, shown in Figure 8-14.



Required parameters

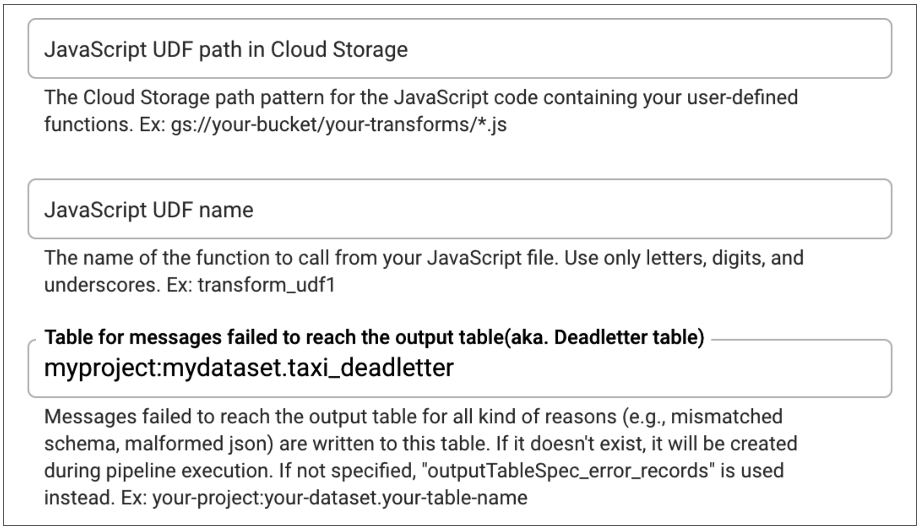
Input Pub/Sub topic *
projects/pubsub-public-data/topics/taxirides-realtime
The Pub/Sub topic to read the input from. Ex: projects/your-project-id/topics/your-topic-name

BigQuery output table *
myproject:mydataset.taxi_data
The location of the BigQuery table to write the output to. If you reuse an existing table, it will be overwritten. The table's schema must match the input JSON objects. Ex: your-project:your-dataset.your-table

Temporary location *
gs://mybucket/temp
Path and filename prefix for writing temporary files. Ex: gs://your-bucket/temp

Figure 8-14. Job parameters

8. Optionally, click SHOW OPTIONAL PARAMETERS and fill in the dead-letter BigQuery table. Take note, in Figure 8-15, of other parameters that are available.



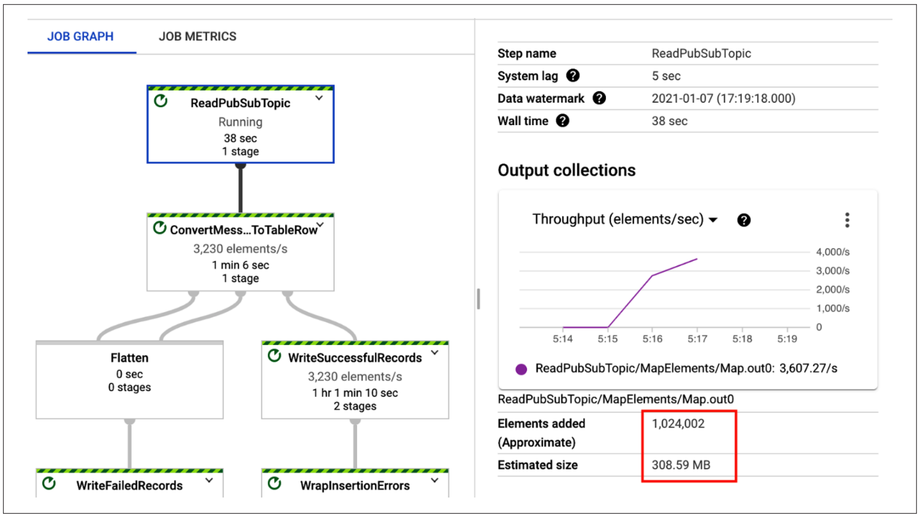
JavaScript UDF path in Cloud Storage
The Cloud Storage path pattern for the JavaScript code containing your user-defined functions. Ex: gs://your-bucket/your-transforms/*.js

JavaScript UDF name
The name of the function to call from your JavaScript file. Use only letters, digits, and underscores. Ex: transform_udf1

Table for messages failed to reach the output table(aka. Deadletter table)
myproject:mydataset.taxi_deadletter
Messages failed to reach the output table for all kind of reasons (e.g., mismatched schema, malformed json) are written to this table. If it doesn't exist, it will be created during pipeline execution. If not specified, "outputTableSpec_error_records" is used instead. Ex: your-project:your-dataset.your-table-name

Figure 8-15. Optional job parameters

9. Click RUN JOB.
10. Give the job a couple of minutes to spin up and then click the ReadPubSubTopic and WriteSuccessfulRecords graph nodes to see that elements are flowing through the dataflow job, as shown in Figure 8-16.



Step name	ReadPubSubTopic
System lag	5 sec
Data watermark	2021-01-07 (17:19:18.000)
Wall time	38 sec

Output collections

ReadPubSubTopic/MapElements/Map.out0	
Elements added (Approximate)	1,024,002
Estimated size	308.59 MB

Figure 8-16. Dataflow job graph and metrics

11. Validate that data is being written to BigQuery from the CLI:

```
bq head mydataset.taxi_data
```

12. Make sure to stop the dataflow job.

Discussion

This recipe shows how to get up and running with your first streaming pipeline quickly. You can extend the functionality to Avro data, using the Avro template. You can also provide a JavaScript transform in the additional parameters for lightweight transformations that doesn't require you to write the entire pipeline code. Any records that fail to be transformed or processed will be written to the dead-letter table for later analysis. (You can test this if you have your own topic and, if you publish a malformed message, it should show up quickly in this table.) Last, if you inspect the details of the BigQuery table in the UI, you'll note that some of your data has been committed to the streaming buffer but is still immediately available for queries.

CHAPTER 9

Data Processing Tools

Google Cloud offers a variety of scalable data processing tools. Dataflow and Dataproc are the most commonly used (outside of BigQuery, covered in another chapter). These tools allow you to run open source Apache Spark or Apache Beam pipelines in a serverless or near-serverless environment. Cloud Dataflow, in particular, is an excellent environment for running large-scale, mission-critical, streaming pipelines for real-time analytics, data ingestion, and business logic. There are also low and no-code data processing toolsets, such as Cloud Data Fusion. These recipes are examples of some of the most common tasks you'll perform as you implement solutions on these tools and include a few more advanced Dataflow pipeline patterns.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

9.1 Cleaning Data Using the Data Fusion GUI

Problem

You want to clean and join data sets in a repeatable pipeline in a low or no-code, GUI-driven tool.

Solution

Cloud Data Fusion allows users to interact with data from sources such as GCS and BigQuery and author-repeatable pipelines in a GUI, and execute them scalably and on a schedule, using Dataproc under the hood.

In this example, we will ingest some data from CSV to BigQuery, applying transformations and filters along the way.

1. From the Cloud Console, navigate to the Data Fusion page. You may need to enable the API, as shown in Figure 9-1.

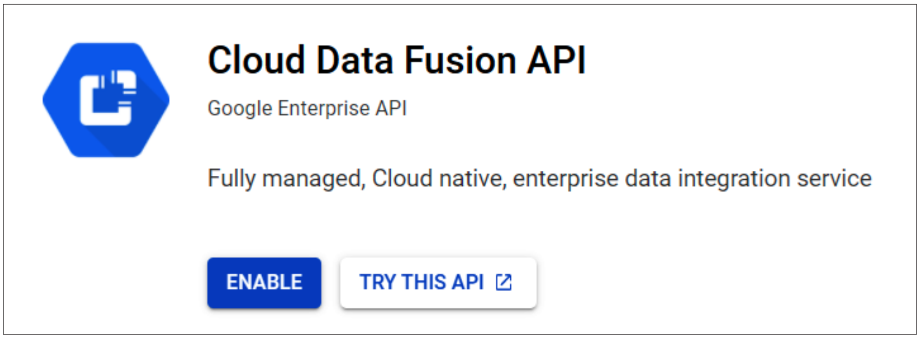

Cloud Data Fusion API
Google Enterprise API
Fully managed, Cloud native, enterprise data integration service
ENABLE
TRY THIS API

Figure 9-1. Enabling the Data Fusion API

2. Once it is enabled, click Create Instance. Name it **`my-data-fusion-instance`**, choose Developer Edition, and grant the requested Dataproc access. It may take 10–20 minutes to create.
3. In the meantime, we'll create a cloud storage bucket and upload a file to process from this recipe's folder in the repo. From the command line, run the following:

```
BUCKET_NAME=gs://<your_project_id>
gsutil mb -l us $BUCKET_NAME
gsutil cp events.csv $BUCKET_NAME
```

4. Once your Data Fusion is created, click View Instance to open Data Fusion.
5. Click the Wrangler section.
6. Open your sample bucket and choose the *events.csv* file, as shown in Figure 9-2.

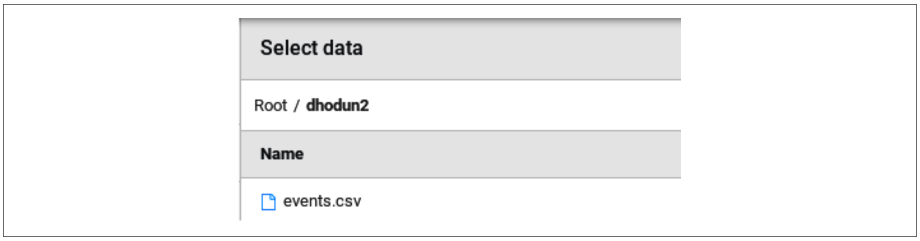


Figure 9-2. The provided events.csv file to wrangle in your pipeline

7. From the Body field drop-down menu, choose Parse and CSV, as shown in Figure 9-3. Choose Comma-Delimited.

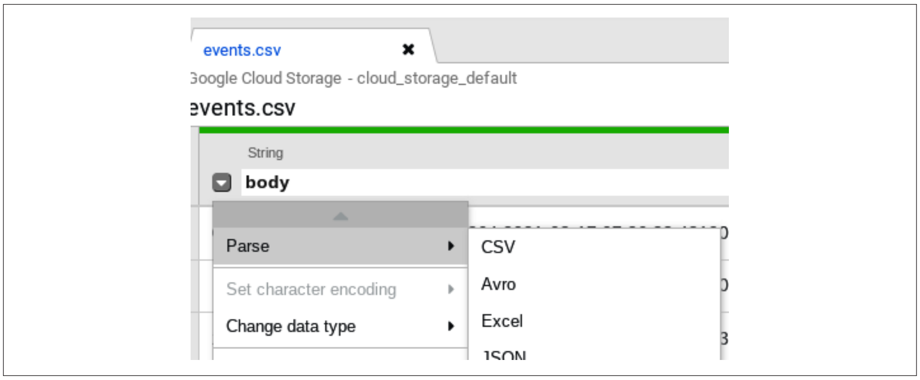


Figure 9-3. Parsing the CSV

8. Since the body column is no longer needed, click the column drop-down menu for Body and then choose Delete Column.
9. You'll notice the fields on the right have now been renamed body, body1, body2, and so on. Click Column Names and then choose Set All to reset the fields quickly to something more meaningful. Type in the following column names, comma-separated, as shown:

   ```
   ip,user_id,lat,lng,time-
   stamp,http_request,http_response,num_bytes,user_agent
   ```

10. Next, we want to filter out page hits on *home.html*, because we are interested in analyzing which of the biology pages are most popular. In the http_request column, click the drop-down menu and choose Filter. Choose Remove Rows, and select If Value Contains. Copy and paste *home.html*, as shown in Figure 9-4. Click Apply.

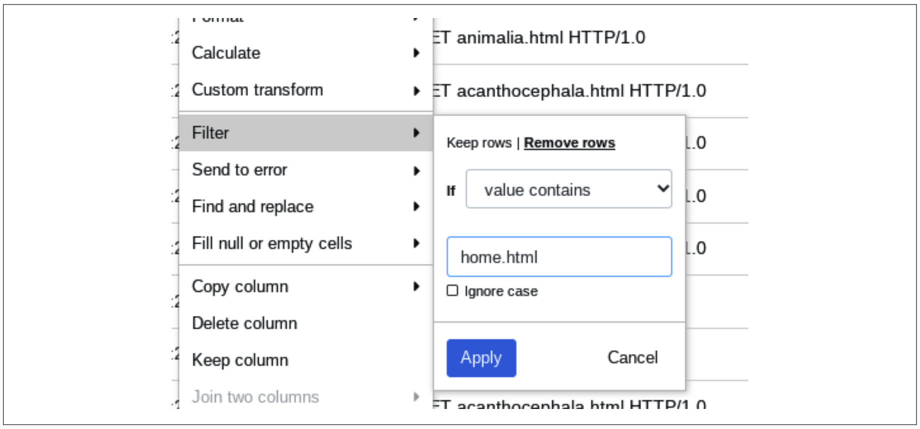
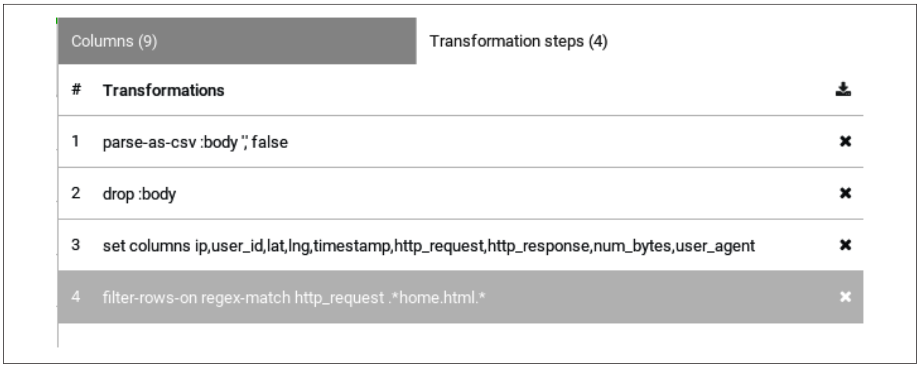


Figure 9-4. Adding a regex filter to our http_request column

11. You will notice that not only is the data displayed being filtered in real time, it is also a series of steps that defines how our pipeline is growing on the right side of the screen, as in Figure 9-5.



#	Transformations
1	parse-as-csv :body ',' false
2	drop :body
3	set columns ip,user_id,lat,lng,timestamp,http_request,http_response,num_bytes,user_agent
4	filter-rows-on regex-match http_request .*home.html.*

Figure 9-5. Our growing pipeline

12. We also don't need the num_bytes column, so drop that as well.
13. Now we are ready to run our simple pipeline. Click Create A Pipeline and choose Batch Pipeline.
14. Name your pipeline **`Weblog-Ingest`**. You can drag and drop the two nodes in our pipeline that you see in Figure 9-6. You can also add more transformations from the left side of the screen or double-click the existing nodes to see their configurations.

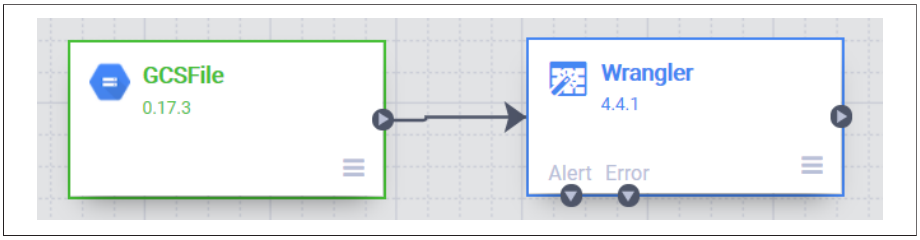


Figure 9-6. The two nodes of our current pipeline

15. We want to store the output in BigQuery, so on the left side, click Sink and then BigQuery. This will create a new node on the graph. Drag a link from the Play button on the Wrangler node to connect it to the new BigQuery node, as shown in Figure 9-7.

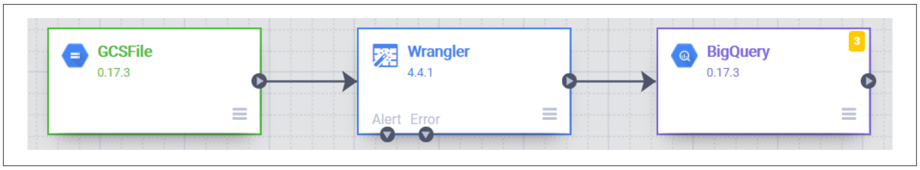


Figure 9-7. Adding a BigQuery sink

16. You'll notice the yellow error number on the BigQuery card. Double-click it to configure this step. Add the following info to the configuration as in the following table:

Field	Value
Reference Name	WebLog-BigQuery-Sink
Project ID	auto-detect
Dataset Project ID	<blank>
Dataset	weblog
Table	filtered_log

17. Click Validate. If no errors are found, click the X to save the step and return to the graph.
18. In the upper right, click Deploy. Once this completes, click Run. You'll see a run status bar change to Provisioning, as in Figure 9-8. Wait for the run to complete.

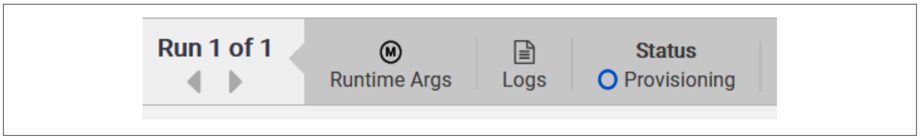


Figure 9-8. The Data Fusion pipeline is provisioning

19. [Optional] If you are curious, you can head to the Dataproc page in the Cloud Console to see the temporary cluster that Data Fusion is creating, using, and then deleting on your behalf.
20. When the status says Succeeded, open the BigQuery page of the Cloud Console. Run the following query to see that your data has been ingested:

```
SELECT * FROM `.weblog.filtered_log`
LIMIT 100
```

21. Don't forget to delete your Data Fusion instance to avoid any long-term charges.

Discussion

Data Fusion is a scalable, GUI-driven data wrangler and ELT/ETL pipeline authoring tool. It provides 150+ preconfigured connectors, includes data lineage, and can even author streaming pipelines. Even though these are no-code pipelines, they run scalably on Dataproc Spark clusters. This recipe showed how easy it is to navigate across disparate data sources and services (i.e., Cloud Storage and BigQuery) and perform transformations on your data.

9.2 Running a Simple Python Dataflow Pipeline

Problem

You want to get started writing Apache Beam pipelines in Python to execute scalable data analytics, image processing, or machine learning preprocessing.

Solution

This recipe will show you how to set up your first Python-based Dataflow pipeline.

1. Open Cloud Shell or your CLI and use Python pip to install the necessary Apache Beam and Dataflow packages. Note the `[gcp]`, which indicates to the pip installer utility to install the additional components needed for Google Cloud. First, we upgrade pip in case you have an older version that can't detect prebuilt binary packages and will therefore try to build them from scratch.

    ```
    pip3 install --upgrade pip
    pip3 install apache-beam[gcp]
    ```

2. Copy the following code into a Python file, either in the Cloud Shell editor or in a text editor, and save it locally as *wordcount.py*. Alternatively, grab a copy from the book's repo. This is a small sample Dataflow pipeline that ships with the framework. It will read from a publicly accessible copy of Shakespeare's *King Lear* and count the frequency of each word. The main pipeline is defined at the lines `lines = p | 'Read' >> ReadFromText(known_args.input)` and in the lines that follow:

    ```
    import argparse
    import logging
    import re

    import apache_beam as beam
    from apache_beam.io import ReadFromText
    from apache_beam.io import WriteToText
    from apache_beam.options.pipeline_options import PipelineOptions
    from apache_beam.options.pipeline_options import SetupOptions

    class WordExtractingDoFn(beam.DoFn):
    ```

```
  """Parse each line of input text into words."""
  def process(self, element):
    return re.findall(r'[\w\']+', element, re.UNICODE)

def run(argv=None, save_main_session=True):
  """Main entry point; defines and runs the wordcount pipeline."""
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--input',
      dest='input',
      default='gs://dataflow-samples/shakespeare/kinglear.txt',
      help='Input file to process.')
  parser.add_argument(
      '--output',
      dest='output',
      required=True,
      help='Output file to write results to.')
  known_args, pipeline_args = parser.parse_known_args(argv)

  # We use the save_main_session option because one or more DoFn's in
  # this workflow relies on global context (e.g., a module imported at
    module level).

  pipeline_options = PipelineOptions(pipeline_args)
  pipeline_options.view_as(SetupOptions).save_main_session =
save_main_session

  # The pipeline will be run on exiting the with block.
  with beam.Pipeline(options=pipeline_options) as p:

    # Read the text file[pattern] into a PCollection.
    lines = p | 'Read' >> ReadFromText(known_args.input)

    counts = (
        lines
        | 'Split' >> (beam.ParDo(WordExtractingDoFn()).with_out-
put_types(str))
        | 'PairWIthOne' >> beam.Map(lambda x: (x, 1))
        | 'GroupAndSum' >> beam.CombinePerKey(sum))

    # Format the counts into a PCollection of strings.
    def format_result(word, count):
      return '%s: %d' % (word, count)

    output = counts | 'Format' >> beam.MapTuple(format_result)

    # Write the output using a "Write" transform that has side effects.
    output | 'Write' >> WriteToText(known_args.output)

if __name__ == '__main__':
  logging.getLogger().setLevel(logging.INFO)
  run()
```

3. Run this Apache Beam pipeline locally on your command line, giving it an output file parameter:

   ```
   python3 wordcount.py --output outputs
   ```

4. Inspect some of the outputs to ensure that the pipeline ran correctly:

   ```
   cat outputs-00000-of-00001
   ```

5. Run this pipeline, using the Dataflow service instead of locally. First, enable the Dataflow API if you haven't already. Run this on CLI.

   ```
   gcloud services enable dataflow
   ```

6. Make a bucket that our pipeline can use as output and to stage temporary files. Run this and replace this bucket name with a new valid bucket name:

   ```
   gsutil mb -l us-central1 gs://my_new_dataflow_bucket23
   ```

7. We'll also need to give the compute service account access to this bucket. This is the account that each Dataflow worker will be running as and will be using to authenticate to other Google Cloud resources. Run the following, replacing [PROJECT_NUMBER] with yours. You can also look up the name of this precreated account by opening the IAM & Admin menu and clicking Service Accounts.

   ```
   PROJECT_ID=<YOUR_PROJECT>
   gcloud projects add-iam-policy-binding $PROJECT_ID \
       --member=serviceAccount:[PROJECT_NUMBER]-
   compute@developer.gserviceaccount.com \
       --role=roles/storage.objectAdmin
   ```

8. Run the following block of code to start your pipeline in Dataflow mode. Make sure to insert your project and bucket in the variables.

   ```
   DATAFLOW_REGION=us-central1
   STORAGE_BUCKET=<YOUR_BUCKET>
   PROJECT_ID=<YOUR_PROJECT>

   python3 wordcount.py \
       --region $DATAFLOW_REGION \
       --input gs://dataflow-samples/shakespeare/kinglear.txt \
       --output gs://$STORAGE_BUCKET/results/outputs \
       --runner DataflowRunner \
       --project $PROJECT_ID \
       --temp_location gs://$STORAGE_BUCKET/tmp/
   ```

9. You should see output confirming that the pipeline has started and there are no permission errors on the bucket. You can also navigate to the Dataflow page in the console and click your new job to see the current status of the pipeline and the steps executing, as in Figure 9-9.

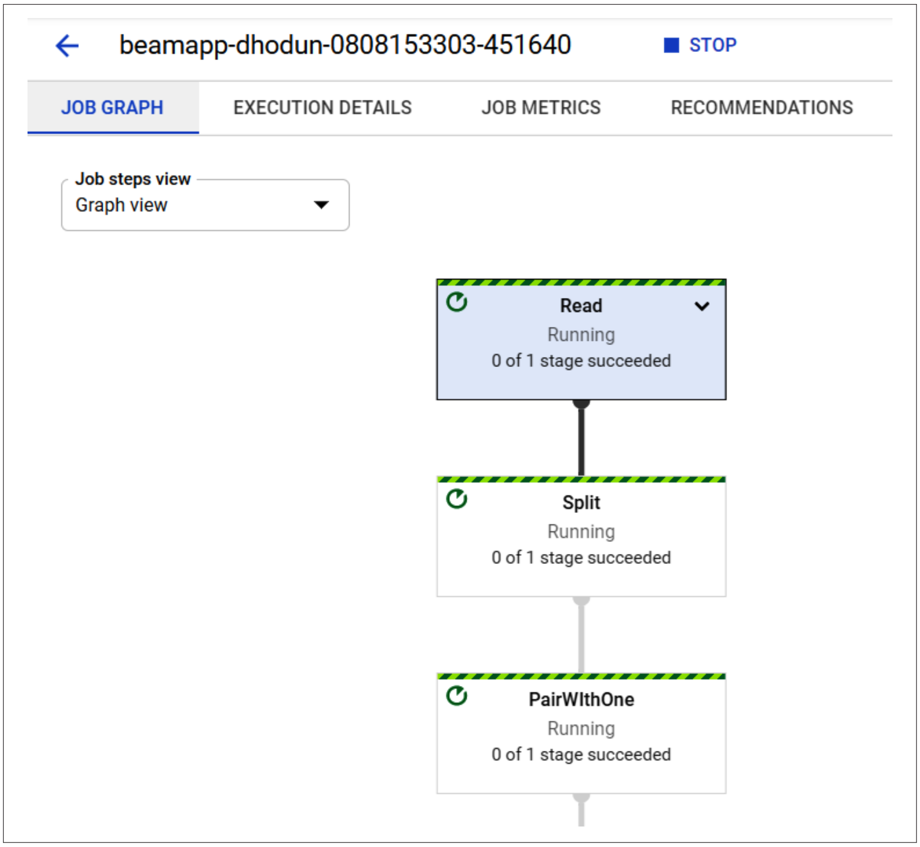


Figure 9-9. Our first Dataflow pipeline

10. When the pipeline has finished executing, you can inspect the files that have been generated by our workers in GCS, just like we had computed locally:

    ```
    gsutil ls gs://$STORAGE_BUCKET/results/
    gsutil cat gs://$STORAGE_BUCKET/results/outputs-00000-of-00003
    ```

Discussion

Dataflow is a serverless, scalable data processing service that executes both batch and streaming Apache Beam pipelines. Beam pipelines can be written in a variety of languages, including Java and Python, and then executed on a variety of runners, including Dataflow, Spark, and Flink. This example shows how to run your first pipeline, both locally and then serverlessly on the Cloud Dataflow services. You enabled the service and configured some additional permissions so that Dataflow workers could access data sources, just like you could from the command line with your user account.

9.3 Building a Streaming Pipeline in Dataflow SQL

Problem

You want to build a streaming pipeline using various Pub/Sub or BigQuery sources, but you want to prototype rapidly and avoid writing a Python or Java Apache Beam pipeline to execute on Dataflow.

Solution

Dataflow SQL allows you to author pipelines purely in SQL and execute them from the Dataflow UI.

To demonstrate this, we will use a sample data set of weblog entries, including an enrichment table with user profile information. The source data is provided in this book's GitHub repository. The user profile contains additional data, with which we want to annotate our weblog to power future analytics and dashboards.

1. First, create the Pub/Sub topic from the CLI or Cloud Shell. We will populate it with data later.

   ```
   PROJECT_ID=<my_project>

   gcloud pubsub topics create events
   ```

2. Create a BigQuery data set and upload data, using the file provided in the repo:

   ```
   bq mk --location=us weblog
   bq load --autodetect $PROJECT_ID:weblog.userprofile ./bq_data.csv
   ```

3. Examine the data to ensure that it's uploaded. Run this `bq` command to return just 10 results from your table:

   ```
   bq head -n 10 $PROJECT_ID:weblog.userprofile
   ```

4. Next, you'll need to add a schema to your Pub/Sub topic so it can be parsed and processed. Create a file called *schema.yaml* and enter the following, or grab it from the repo:

   ```
   - column: event_timestamp
     description: Pub/Sub event timestamp
     mode: REQUIRED
     type: TIMESTAMP
   - column: ip
     description: IP Address
     mode: NULLABLE
     type: STRING
   - column: user_id
     description: User ID
     mode: NULLABLE
     type: STRING
   - column: lat
   ```

```
    description: Latitude
    mode: NULLABLE
    type: FLOAT64
  - column: lng
    description: Longitude
    mode: NULLABLE
    type: FLOAT64
  - column: timestamp
    description: Event timestamp
    mode: NULLABLE
    type: STRING
  - column: http_request
    description: HTTP Request
    mode: NULLABLE
    type: STRING
  - column: http_response
    description: HTTP Response Code
    mode: NULLABLE
    type: INT64
  - column: num_bytes
    description: Number response bytes
    mode: NULLABLE
    type: INT64
  - column: user_agent
    description: User Browser
    mode: NULLABLE
    type: STRING
```

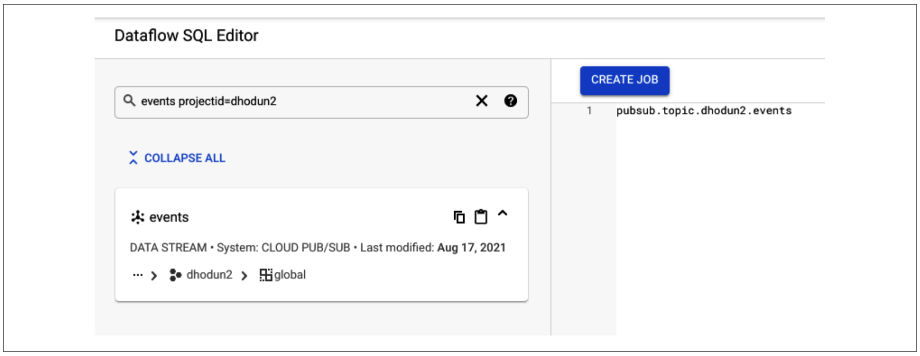
5. Now add this schema to data catalog so that Dataflow SQL will know how to parse the incoming messages:

```
gcloud data-catalog entries update --lookup-entry='pubsub.topic.
$PROJECT_ID.events' \
    --schema-from-file=schema.yaml
```

6. Generate Pub/Sub data and publish to your topic for processing, using your CLI or Cloud Shell. This script is provided in the repo and simulates the data stream from another team that you want to consume.

```
export PROJECT_ID=$PROJECT_ID
python publish_pubsub.py
```

7. In the Dataflow UI, click Create Job From SQL. In the data set search bar, look for your events topic, as in Figure 9-10. You can click the Paste button to paste your topic into the SQL editor properly.



DATA STREAM • System: CLOUD PUB/SUB • Last modified: Aug 17, 2021

··· > dhodun2 > global

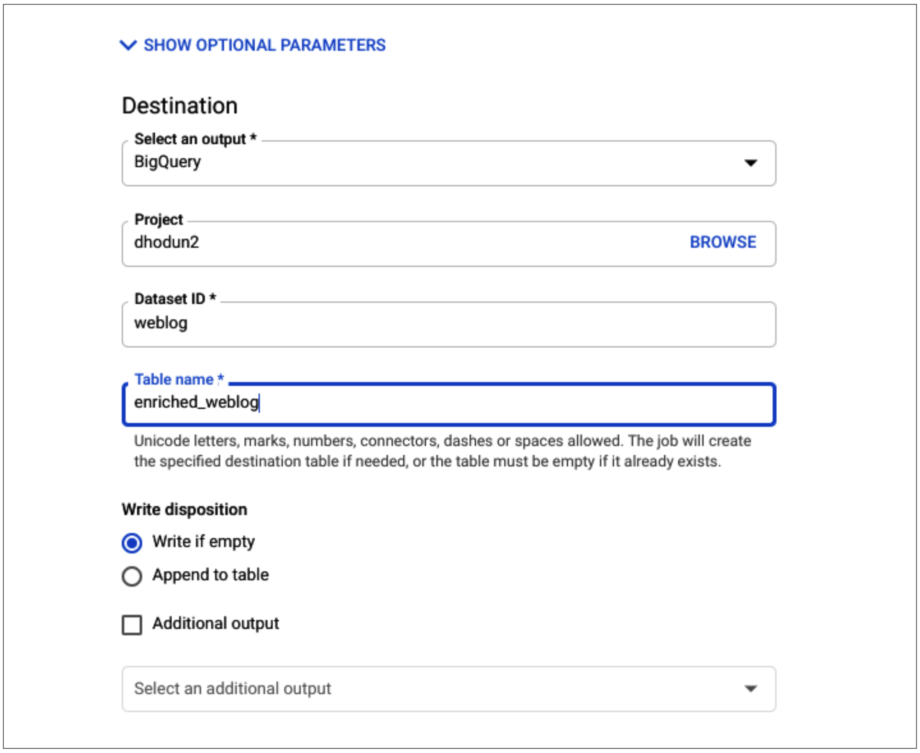
```
pubsub.topic.dhodun2.events
```

Figure 9-10. Dataflow SQL data set search

8. Now you can author a fully streaming pipeline in the Dataflow SQL UI. Copy and paste the following from the repo and update your project name. This query demonstrates joining a BigQuery enrichment table with a real-time Pub/Sub stream.

```
SELECT
  events.event_timestamp,
  events.ip,
  events.user_id,
  events.http_request,
  table.age_bracket,
  table.opted_into_marketing,
  table.last_visit
FROM
  pubsub.topic.dhodun2.events AS events
INNER JOIN
  bigquery.table.dhodun2.weblog.userprofile table
ON
  events.user_id = table.id
```

9. Now click Create Cloud Dataflow Job. Be sure to add an output BigQuery source, as shown in Figure 9-11, and then click Create. You can click the Job_Id link to look at the job status and the graph that has been created from your SQL.



Destination

Select an output *
BigQuery

Project
dhodun2

Dataset ID *
weblog

Table name *
enriched_weblog

Unicode letters, marks, numbers, connectors, dashes or spaces allowed. The job will create the specified destination table if needed, or the table must be empty if it already exists.

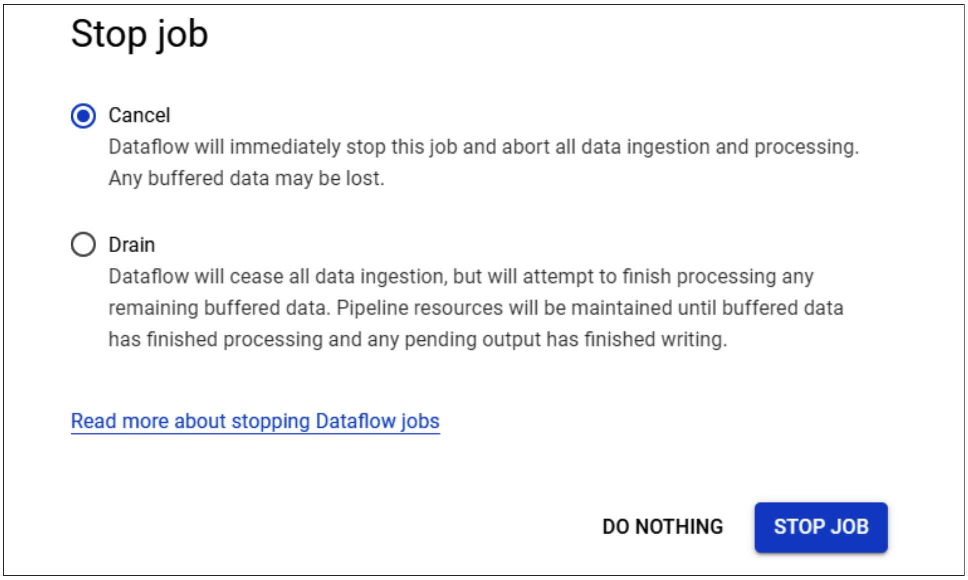
Write disposition
[✓ Write if empty]
Append to table

Additional output

Select an additional output

Figure 9-11. Adding a BigQuery output table

10. Once the table has run for 3–5 minutes, take a look at weblog.enriched_weblog in the BigQuery UI. You'll notice that not only are we taking data from a Pub/Sub topic and writing it to BigQuery with SQL, which is quite helpful, we are also joining this data to a BigQuery data set.
11. Since this is a streaming job, it will execute forever, so don't forget to cancel your job, as in Figure 9-12, and close the process that was generating events.


Stop job

[✓ Cancel]
Dataflow will immediately stop this job and abort all data ingestion and processing. Any buffered data may be lost.

Drain
Dataflow will cease all data ingestion, but will attempt to finish processing any remaining buffered data. Pipeline resources will be maintained until buffered data has finished processing and any pending output has finished writing.

Read more about stopping Dataflow jobs

Figure 9-12. Canceling your streaming Dataflow job

Discussion

Dataflow SQL is an extension of Apache Beam SQL, which allows users to write simple (or complex) streaming pipelines in SQL from the Cloud Console. This example shows how to join an unbounded Pub/Sub data set with a bounded BigQuery enrichment data set. We stored this data in BigQuery, but we could have just as easily written it to a new Pub/Sub topic. Dataflow also allows you to perform aggregate functions on time slices of your data, using SQL Windowing functions `TUMBLE`, `HOP`, and `SESSION` for fixed, sliding, and session-based windows, respectively.

9.4 Querying BigQuery from a Dataproc Job

Problem

You need to configure your Spark data processing job to access source data in BigQuery, your data warehouse.

Solution

Using initialization actions when you create your Dataproc clusters, you can easily enable access to BigQuery, Google Cloud Storage, and other storage systems external to your Dataproc cluster.

1. First, create a Dataproc cluster with the necessary connectors and initialization actions. This leverages a Google-provided initialization script to your cluster nodes that installs GCS and BigQuery connectors so you can query data from these services. You can configure these as needed.

2. Run this from the CLI or Cloud Shell:

```
REGION=us-central1
CLUSTER_NAME=sample-cluster
gcloud dataproc clusters create ${CLUSTER_NAME} \
    --region ${REGION} \
    --initialization-actions gs://goog-dataproc-initialization-actions-$
{REGION}/connectors/connectors.sh \
    --metadata gcs-connector-version=2.2.0 \
    --metadata bigquery-connector-version=1.2.0 \
    --metadata spark-bigquery-connector-version=0.19.1
```

3. While waiting for the cluster to create, upload a table into BigQuery that we will query, using the file from the repo:

```
bq load -location us --autodetect $PROJECT_ID:weblog.raw_events
events_with_header.csv
```

4. Prepare a PySpark script to submit that will load an entire table into Spark for additional querying. Copy this code into the Cloud Shell editor, a text editor, or your favorite IDE. You can also find it in this book's GitHub repo. This script will read data from your BigQuery raw_events table, perform some aggregations based on user_id, and write it out to a new events_aggregate table.

```
#!/usr/bin/python
from pyspark.sql import SparkSession

PROJECT = "PROJECT_ID"

spark = SparkSession \
  .builder \
  .master('yarn') \
  .appName('cookbook-query') \
  .getOrCreate()

# Use the Cloud Storage bucket for temporary BigQuery export data used
# by the connector.
bucket = PROJECT
spark.conf.set('temporaryGcsBucket', bucket)

# Load data from BigQuery.
events = spark.read.format('bigquery') \
  .option('table', f'{PROJECT}:weblog.raw_events') \
  .load()
events.createOrReplaceTempView('raw_events')

# Count how many events per user
```

```
regions_count = spark.sql(
    'SELECT COUNT(*) AS num_visits, user_id FROM raw_events GROUP BY
user_id')
regions_count.show()
regions_count.printSchema()

# Saving the data to BigQuery
regions_count.write.format('bigquery') \
  .option('table', 'weblog.events_aggregate') \
  .save()
```

5. Last, from the CLI, submit the query using the Dataproc Jobs API. Notice that you are using a prebuilt jar and providing the script it will execute.

```
#!/bin/bash

REGION=us-central1
CLUSTER_NAME=sample-cluster

gcloud dataproc jobs submit pyspark query_script.py \
    --cluster=$CLUSTER_NAME \
    --region=$REGION \
    --jars=gs://spark-lib/bigquery/spark-bigquery-latest.jar
```

6. You should see job output and completion on your CLI. You can also check the BigQuery UI to see a new table called sales_region_aggregate in your data set.
7. Delete your cluster to avoid charges:

```
CLUSTER_NAME=sample-cluster
gcloud dataproc clusters delete ${CLUSTER_NAME}
```

Discussion

Dataproc is Google Cloud's managed Spark service and allows for efficient, scalable execution of your Spark pipelines, using just-in-time clusters provisioned in 90 seconds. Dataproc has a number of easy-to-use connectors for accessing your Google Cloud data, primarily the BigQuery and Google Cloud Storage connectors. This allows you to write pipelines in OSS Spark code but easily offload the storage (and sometimes, in the case of BigQuery, significant compute) from the cluster into cloud-native, cheaper, and more efficient solutions. This allows for the paradigm of per-job clusters or short-lived clusters versus a long-lived, monolithic cluster which adds operational burden.

9.5 Adding Event Timestamps to Pub/Sub

Problem

The `timestamp` attribute on a Pub/Sub message indicates the publish time, that is, when the message reached the Pub/Sub service and not necessarily when the event actually was created, particularly in offline scenarios (such as when your mobile phone is in a tunnel or in airplane mode). You want to process based on when the event actually happened, not the publish time from the view of the Pub/Sub service, which is the default.

Solution

You can add metadata to Pub/Sub messages, which can then be consumed by downstream applications. This example shows how to do it in Python:

1. Generate a timestamp as part of your payload dictionary. You can follow along with the entire script in the GitHub repo.

   ```
   def generate_event():
       return {
           'user_id': random.randint(1, 100),
           'action': random.choices(['start', 'stop', 'rewind',
             'download'], k=1),
           'timestamp': datetime.datetime.utcnow().strftime(
             '%m/%d/%Y %H:%M:%S %Z')
       }
   ```

2. When you publish to Pub/Sub, add a `timestamp` field in addition to the encoded data:

   ```
   publisher = pubsub_v1.PublisherClient()
   topic_path = publisher.topic_path(project_id, topic_name)

   buffer = []

   def publish_burst(publisher, topic_path, buffer):
       for message in buffer:
           json_str = json.dumps(message)
           data = json_str.encode('utf-8')
           publisher.publish(topic_path, data,
                             timestamp=message['timestamp'])
           print('Message for event {} published at {}'.format(
               message['timestamp'], datetime.datetime.utcnow().
               strftime('%m/%d/%Y %H:%M:%S %Z')))
   ```

3. Now your Pub/Sub messages will have an event `timestamp` attribute you can access in addition to the automatically set `publishTime` metadata. You can now use the `timestamp` value to order your data based on when it occurs. Cloud

Dataflow, for example, allows you to use `event_timestamp` in lieu of the publish time with the following (in Java). By default, Dataflow would just use publish time.

```
pipeline.apply("ReadMessage", PubsubIO.readStrings()
                        .withTimestampAttribute("timestamp")
                        .fromTopic(options.getInputTopic()))
```

4. Run the sample script by running the following and examine the difference between event time and publish time. These are close in value, but the difference could be significant:

```
python3 publish.py

You should see something like this:
08/30/2021 19:22:22
Message for event 08/30/2021 19:22:18  published at 08/30/2021 19:22:22
Message for event 08/30/2021 19:22:19  published at 08/30/2021 19:22:22
Message for event 08/30/2021 19:22:20  published at 08/30/2021 19:22:22
Message for event 08/30/2021 19:22:21  published at 08/30/2021 19:22:22
Message for event 08/30/2021 19:22:22  published at 08/30/2021 19:22:22
Message for event 08/30/2021 19:22:28  published at 08/30/2021 19:22:37
Message for event 08/30/2021 19:22:29  published at 08/30/2021 19:22:37
Message for event 08/30/2021 19:22:30  published at 08/30/2021 19:22:37
Message for event 08/30/2021 19:22:31  published at 08/30/2021 19:22:37
Message for event 08/30/2021 19:22:32  published at 08/30/2021 19:22:37
```

Discussion

You can set attributes in Pub/Sub messages that can be intelligently consumed in downstream data, in this case, a `timestamp` attribute. Dataflow is smart enough not only to use this timestamp in place of publishing timestamp for time-based processing such as windowing, but it can also use this metadata efficiently to update its watermark—the internal mechanism by which it tracks how up to date in-flight data is. A lagging watermark signals Dataflow to add more workers to process data more quickly.

9.6 Inferring and Using Schemas in Dataflow

Problem

Your current Dataflow pipeline leverages custom coders, even though your data is well-structured and defined in plain old Java objects (POJOs). In addition, you have to write out the schema of the objects for `BigQueryIO.Write()`. You want to write more concise, readable Dataflow pipelines and avoid custom coders to speed up development.

Solution

Apache Beam gives you the ability to define or infer schemas from well-structured data, which automatically gives you high-performance encoding and automatically creates the BigQuery schema.

This recipe and the next few are more advanced examples showing newer Dataflow features. They assume knowledge of Java and of building basic Dataflow pipelines. If you're new to Dataflow, you can still follow along to learn some new concepts. You will need Java and Maven installed. Cloud Shell is a good environment with these already set up.

1. Navigate to the example folder in the repo. Open the example pipeline file to understand the code (*src/main/java/com/mypackage/pipeline/MyPipeline.java*).
2. First, create a schema for any objects that Beam will need to pass between stages of the pipeline. Note that you can mark fields as nullable. In this case, we have created the schema for the same log object used in the previous examples. You can see this already implemented in `MyPipeline.java`:

```
public static Schema CommonLog = Schema.builder()
        .addStringField("user_id")
        .addStringField("ip")
        .addNullableField("lat", Schema.FieldType.DOUBLE)
        .addNullableField("lng", Schema.FieldType.DOUBLE)
        .addStringField("timestamp")
        .addStringField("http_request")
        .addStringField("user_agent")
        .addInt64Field("http_response")
        .addInt64Field("num_bytes")
        .build();
```

3. Now, you can use this schema in your pipeline to avoid using custom coders. You can also convert to and from `<Row>` objects, which are the generic schematized object in Beam:

```
pipeline.apply("ReadFromPubSub",
             PubsubIO.readStrings().fromTopic(topic))
        .apply("JsonToRow", JsonToRow.withSchema(CommonLog))
```

4. You can now add the `useBeamSchema()` object on your `BigQueryIO.write()` calls, whereas previously you had to create a `TableSchema` object and write out the structure of the data again to pass to `BigQueryIO.write()`:

```
.apply("WriteToBQ",
  BigQueryIO.<Row>write().to(output).useBeamSchema()
    .withWriteDisposition(
        BigQueryIO.Write.WriteDisposition.WRITE_APPEND)
    .withCreateDisposition(
        BigQueryIO.Write.CreateDisposition.CREATE_IF_NEEDED));
```

5. To run the example pipeline, do the following in the example directory in Cloud Shell or on the CLI. First, start the Python generator that publishes data to Pub/Sub:

   ```
   python publish_pubsub.py
   ```

6. Open a new tab so you can leave the data generator running. Replace your project variables here in the *//static inputs and outputs* section and execute the following in the recipe folder to compile and run your pipeline. You can also find this in the *run.sh* file:

   ```
   # Set up environment variables
   export PROJECT_ID=$(gcloud config get-value project)
   export REGION='us-central1'
   export PIPELINE_FOLDER=gs://${PROJECT_ID}
   export MAIN_CLASS_NAME=com.mypackage.pipeline.MyPipeline
   export RUNNER=DataflowRunner

   mvn compile exec:java \
   -Dexec.mainClass=${MAIN_CLASS_NAME} \
   -Dexec.cleanupDaemonThreads=false \
   -Dexec.args=" \
   --project=${PROJECT_ID} \
   --region=${REGION} \
   --stagingLocation=${PIPELINE_FOLDER}/staging \
   --tempLocation=${PIPELINE_FOLDER}/temp \
   --runner=${RUNNER}"
   ```

7. That command will run to completion, but the job is still running. You can examine it in the Dataflow UI.
8. Once the job has been running 3–5 minutes, you should see outputs to the *weblog.dataflow_streaming* table.
9. Cancel your streaming pipeline to avoid charges.

Discussion

More traditional Apache Beam pipelines, run on Dataflow, leveraged the <*K,V*> (or *Key, Value*) style of Beam execution. You would pass objects from step to step in the pipeline, extracting a *Key* when it was needed for other aggregations, such as Summing on a GroupByKey. Pipeline authors often had to reason about how to encode these objects, since they needed to be serialized between each pipeline step. Newer Beam schemas allow for a more natural way to encode and address well-structured data. The preceding recipe shows how to define a schema to be used by your pipeline steps. You can also infer a schema from a POJO definition versus writing a custom coder. It also shows how to write this data easily into BigQuery without having to supply another pretty much redundant `TableSchema` object.

9.7 Mini-batching and Streaming Dataflow Data to BigQuery Using Filters

Problem

Your current Dataflow pipeline leverages BigQuery streaming. Since BigQuery charges for streaming inserts, you may want to control your costs if you are processing a large amount of data and only stream a subset of the data.

Solution

Implement BigQuery batch loading in your Dataflow pipeline, potentially on a subset of less time-sensitive data to the same table. Batch loading is great for data that can be loaded every 90 seconds or longer. You can split the data with a branching pipeline and filter functions.

1. Navigate to the example folder in the repo. Open the *example pipeline* file to understand the code (*src/main/java/com/mypackage/pipeline/MyPipeline.java*).
2. First, branch your pipeline the standard way and implement a schema-aware filter function to determine which elements you want persisted in which method:

```
// Streaming Entries
commonLogs.apply("FilterLargeAccounts"),
    Filter.<Row>create().whereFieldName("num_bytes",
                  (Long num_bytes) -> num_bytes > amountCutoff))

// commonLogs Entries
commonLogs.apply("FilterSmallAccounts"),
    Filter.<Row>create().whereFieldName("num_bytes",
                  (Long num_bytes) -> num_bytes > amountCutoff))
```

3. Next, implement your BigQuery write as before on the streaming branch:

```
.apply("WriteToBQStreaming",
  BigQueryIO.<Row>write().to(output).useBeamSchema()
      .withWriteDisposition(
          BigQueryIO.Write.WriteDisposition.WRITE_APPEND)
      .withCreateDisposition(
          BigQueryIO.Write.CreateDisposition.CREATE_IF_NEEDED));
```

4. Implement a Batch_Loads BigQuery writer by setting `.withMethod()`, `.withTriggeringFrequency()`, and `.withNumFileShards()`:

```
.apply("WriteToBQFileLoads",
  BigQueryIO.<Row>write().to(output).useBeamSchema()
      .withMethod(BigQueryIO.Write.Method.FILE_LOADS)
      .withTriggeringFrequency(
        Duration.standardSeconds(minibatchFrequency))
      .withNumFileShards(1)
```

```
        .withWriteDisposition(
          BigQueryIO.Write.WriteDisposition.WRITE_APPEND)
        .withCreateDisposition(BigQueryIO.Write.CreateDisposition.CRE-
ATE_IF_NEEDED));
```

5. Like before, start the Python generator that publishes data to Pub/Sub from CLI:

   ```
   python publish_pubsub.py
   ```

6. Run your pipeline, after you've replaced your project IDs in the code:

   ```
   bash run.sh
   ```

   Then open the pipeline details and you'll see something similar to Figure 9-13.

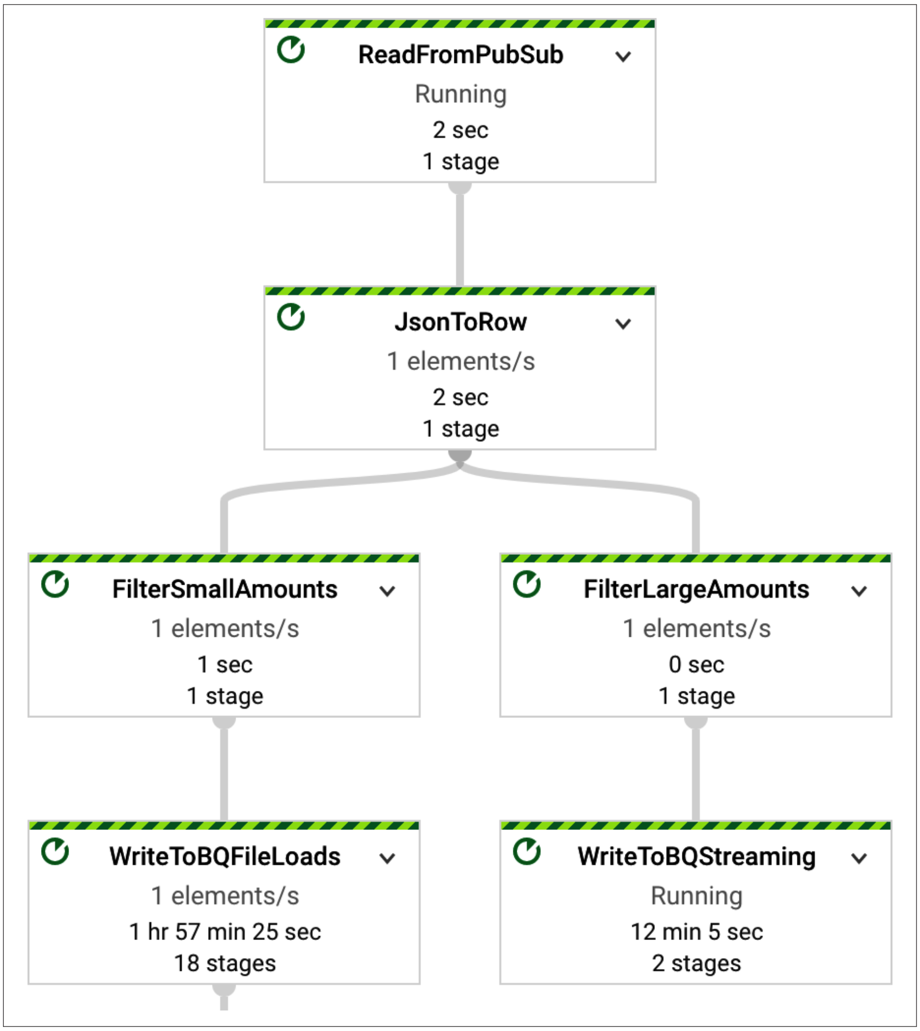


Figure 9-13. Our branched pipeline with both streaming and batch BigQuery loads

7. Once the job has been running 3–5 minutes, you should see outputs to the *weblog.dataflow_streaming* table.
8. Cancel your streaming pipeline to avoid charges.

Discussion

By default, Dataflow writes to BigQuery, using streaming inserts in a streaming pipeline and a batch load job in a batch pipeline. Streaming inserts allow you to query the data immediately, allowing for real-time reporting and dashboards. However, there is an ingestion cost associated with streaming inserts, whereas batch load jobs are free. Generally, best practice is to start with streaming inserts, but if this ingestion cost becomes an issue, you can usually identify some data to offload to "batch" loads with the preceding method. Currently, this is possible only in the Java SDK, not in the Python SDK.

9.8 Triggering a Dataflow Job Automatically from a GCS Upload

Problem

You want to perform an ETL job automatically on flat-file raw data staged into GCS. Another customer or team with which you work will periodically upload these files. You don't want to rely just on time-based scheduling to ingest the data automatically. Rather, you want to perform the ingest job right away.

Solution

You can leverage Dataflow templates and Google Cloud function triggers to start a Dataflow job automatically to process the file(s) immediately when they are landed.

1. As before, navigate to the example folder in the repo. Open the example pipeline file to understand the code, found at *src/main/java/com/mypackage/pipeline/MyPipeline.java*.
2. This time, we parameterize it properly with an `Options` class instead of hard-coding the inputs and outputs:

```
public interface Options extends DataflowPipelineOptions {
  @Description("Path to events.json")
  String getInputPath();

  void setInputPath(String inputPath);

  @Description("Output BigQuery table.")
  String getOutputTable();
```

```
    void setInputPath(String value);
  }

  public static void main(String[] args) {
    Options options = PipelineOptionsFactory.fro-
  mArgs(args).as(Options.class);

    run(options);
  }
```

3. Package the entire pipeline into an Uber *.jar* file. An Uber jar contains all the dependencies required for the pipeline, so it can be shipped and executed as a single unit. Run this from the recipe example directory:

   ```
   mvn package
   ls -lh target/*.jar
   ```

   Now that you have an Uber jar, we will build and stage a flex template, which will capture additional parameters as well as the full executable and package them further in a Docker image. This can then be called at any time, by anyone authorized, to start the pipeline with new parameters—in this case, the file to ingest.

4. First, make sure you have a *metadata.json* file set for your parameters:

   ```
   BUCKET=[BUCKET_NAME]
   PROJECT=[PROJECT_NAME]

   export TEMPLATE_PATH=gs://$BUCKET/tmp/templates
   export TEMPLATE_IMAGE="gcr.io/$PROJECT/cookbook/gcs-trigger-
   pipeline:latest"


   gcloud dataflow flex-template build $TEMPLATE_PATH \
     --image-gcr-path "$TEMPLATE_IMAGE" \
     --sdk-language "JAVA" \
     --flex-template-base-image JAVA11 \
     --metadata-file "metadata.json" \
     --jar "target/cookbook-gcs-trigger-pipeline-1.0.jar" \
     --env FLEX_TEMPLATE_JAVA_MAIN_CLASS="com.mypackage.
       pipeline.MyPipeline"
   ```

5. Copy the data to your bucket for a test run:

   ```
   gsutil cp data.csv gs://$BUCKET/tmp/events.json
   ```

6. Test calling the template from the command line:

   ```
   export TEMPLATE_PATH=gs://$BUCKET/tmp/templates/gcs-trigger-
   template.json
   export TEMPLATE_IMAGE="gcr.io/$PROJECT/cookbook/gcs-trigger-
   pipeline:latest"

   export INPUT_PATH="gs://$BUCKET/tmp/events.json"
   ```

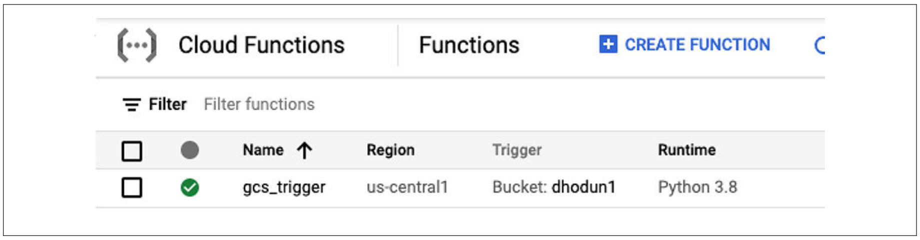
```
export OUTPUT_TABLE=${PROJECT_ID}:weblog.events_upload

export REGION="us-central1"

gcloud dataflow flex-template run "cookbook-trigger-pipeline-`date +%Y%m
%d-%H%M%S`" \
  --template-file-gcs-location "$TEMPLATE_PATH" \
  --parameters inputPath="$INPUT_PATH" \
  --parameters outputTable="$OUTPUT_TABLE" \
  --region "$REGION"
```

7. After a couple of minutes, you'll see your pipeline queued and then started in the Dataflow UI. Once it starts properly, cancel the job.
8. Next, we'll deploy a GCS trigger, using Cloud Functions to trigger the pipeline. Move into the *gcs_trigger* folder and take a look at the code in `main.py`. Update it with your project details. Run the following in this folder to enable Cloud Functions and deploy this trigger:

```
cd gcs_trigger/

gcloud services enable cloudfunctions


gcloud functions deploy gcs_trigger \
--runtime python38 \
--trigger-resource $BUCKET \
--trigger-event google.storage.object.finalize
```

9. Verify in the Cloud Functions UI that your trigger is deployed, as in Figure 9-14.
10. Trigger your new pipeline by re-creating the JSON file. The pipeline is created automatically by the cloud function, the Flex Template kicks off, and eventually the pipeline completes and data is reinserted in BigQuery:

```
gsutil cp events.json gs://$BUCKET/tmp/events.json
```



		Name ↑	Region	Trigger	Runtime
☐	✓	gcs_trigger	us-central1	Bucket: dhodun1	Python 3.8

Figure 9-14. Deployed GCS trigger

11. Delete your cloud function when finished.

Discussion

There are several moving parts here. First is the notion of a fully parameterized, reusable pipeline. You can build more generic pipelines directly in Java with `ValueProvider` classes, but they get a little trickier to author, add extra code to your pipeline, and make it less readable. Flex templates come in handy here and allow you to take more natural parameters on your pipeline. Flex templates also create a consistent deployment environment that any system can trigger. This is less of an issue with Java (since you can point Dataflow to use an Uber jar on GCS), but more important for Python, where you need your triggering environment to have consistent PyPi dependencies. Last, we used a GCF trigger on GCS to interact with the Dataflow API and pass each new file in the bucket to the pipeline.

CHAPTER 10

AI/ML

Machine learning (ML) and AI hold an increasingly important place in enterprise applications. Google Cloud offers a number of AI and ML services, from pre-trained APIs that can be added to existing applications with a few lines of code to the full-featured Vertex AI platform that can be used to train and operationalize ML models in many frameworks.

With model training and tuning becoming more automated, in particular with tools like AutoML, organizations are focusing more on advanced concepts, including continuous retraining and deployment with MLOps, as well as deploying explainable AI in the enterprise. In this chapter, we will present a number of recipes, using Vertex AI, from setting up your customized environment to training and deploying your first model, to more specific techniques aimed at integrating other services. These recipes assume a basic understanding of typical Python data science tools—for example, Jupyter notebooks and the Pandas library.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

10.1 Creating a Vertex AI Notebook

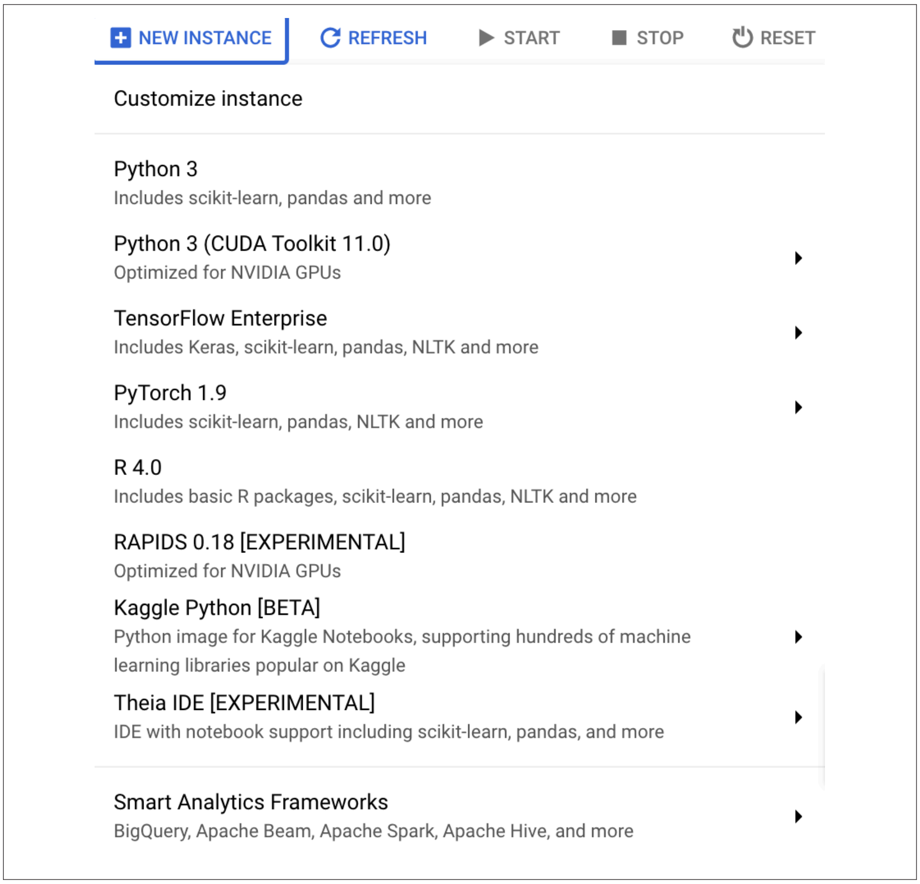
Problem

You need a hosted Jupyter Notebook environment running in Google Cloud that can easily connect to other Google services to perform data and ML tasks.

Solution

You can create, customize, and connect to a Vertex AI notebook.

1. From the Google menu bar, select Vertex AI > Notebooks.
2. Choose New Instance, and you'll see a list of available instances, as shown in Figure 10-1.



Customize instance

Python 3
Includes scikit-learn, pandas and more

Python 3 (CUDA Toolkit 11.0)
Optimized for NVIDIA GPUs

TensorFlow Enterprise
Includes Keras, scikit-learn, pandas, NLTK and more

PyTorch 1.9
Includes scikit-learn, pandas, NLTK and more

R 4.0
Includes basic R packages, scikit-learn, pandas, NLTK and more

RAPIDS 0.18 [EXPERIMENTAL]
Optimized for NVIDIA GPUs

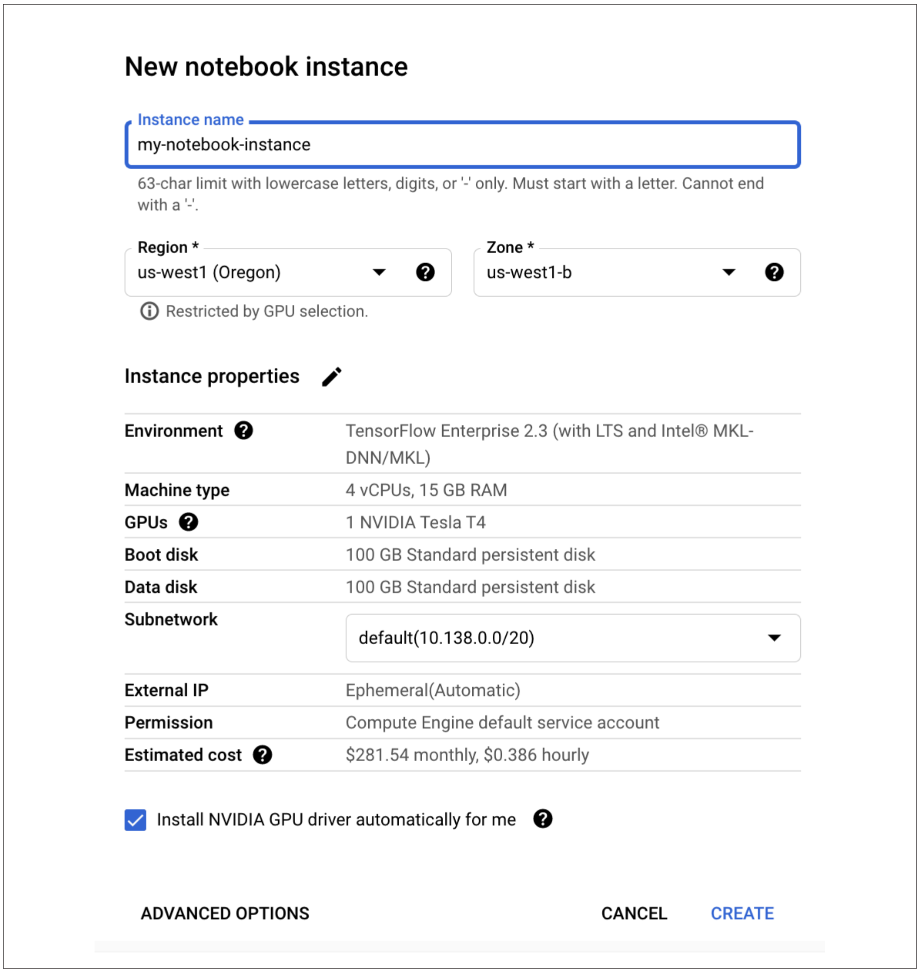
Kaggle Python [BETA]
Python image for Kaggle Notebooks, supporting hundreds of machine learning libraries popular on Kaggle

Theia IDE [EXPERIMENTAL]
IDE with notebook support including scikit-learn, pandas, and more

Smart Analytics Frameworks
BigQuery, Apache Beam, Apache Spark, Apache Hive, and more

Figure 10-1. New Notebook instance dialog box

3. You'll now see instance options, as shown in Figure 10-2. Choose the latest TensorFlow Enterprise option and a GPU to attach if needed. Certain model-training activities perform much faster.
4. Change to **`my-notebook-instance`** and make sure you have selected a four-vCPU machine.
5. Select Install NVIDIA GPU Driver Automatically if you have chosen to use a GPU.



New notebook instance

Instance name: my-notebook-instance

63-char limit with lowercase letters, digits, or '-' only. Must start with a letter. Cannot end with a '-'.

Region *: us-west1 (Oregon)

Zone *: us-west1-b

Restricted by GPU selection.

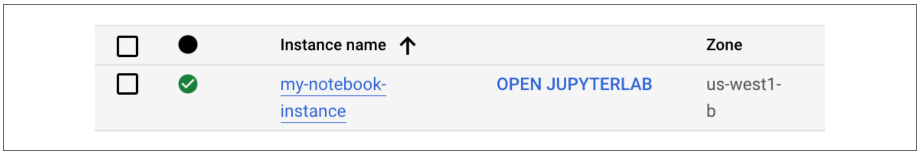
Instance properties

Environment	TensorFlow Enterprise 2.3 (with LTS and Intel® MKL-DNN/MKL)
Machine type	4 vCPUs, 15 GB RAM
GPUs	1 NVIDIA Tesla T4
Boot disk	100 GB Standard persistent disk
Data disk	100 GB Standard persistent disk
Subnetwork	default(10.138.0.0/20)
External IP	Ephemeral(Automatic)
Permission	Compute Engine default service account
Estimated cost	$281.54 monthly, $0.386 hourly

[✓] Install NVIDIA GPU driver automatically for me

Figure 10-2. Customize in the New Notebook Instance dialog box

6. Click Create.
7. You will now see your instance listed; when the status indicators stop spinning and OPEN JUPYTERLAB appears, as in Figure 10-3, click the latter to open your notebook environment.



☐	●	Instance name ↑		Zone
☐	✓	my-notebook-instance	OPEN JUPYTERLAB	us-west1-b

Figure 10-3. Initialized notebook in the UI

8. Alternatively, you can create a notebook instance via the CLI with the following from Cloud Shell or the CLI:

   ```
   export INSTANCE_NAME="example-instance"
   export VM_IMAGE_PROJECT="deeplearning-platform-release"
   export VM_IMAGE_FAMILY="tf2-2-3-cpu"
   export MACHINE_TYPE="n1-standard-4"
   export LOCATION="us-central1-b"

   gcloud notebooks instances create $INSTANCE_NAME \
     --vm-image-project=$VM_IMAGE_PROJECT \
     --vm-image-family=$VM_IMAGE_FAMILY \
     --machine-type=$MACHINE_TYPE --location=$LOCATION
   ```

9. Click OPEN JUPYTERLAB to connect to your notebook environment.
10. Delete the notebook to avoid charges, or leave it to complete other recipes.

Discussion

Vertex AI notebooks are where you will perform much of your data and ML work. The notebooks are a hosted, customizable environment that handles things like installing data science dependencies, installing and configuring NVIDIA drivers (a big win for anyone who has done this more than a couple of times!), handling authentication to Google Cloud APIs—either as a service account or user account—and creating a reverse proxy to connect securely via a browser into the notebook and cloud environment. They can be further configured or locked down for more secure environments; for example, they can be protected by Virtual Private Cloud Service Controls (VPC-SC).

10.2 Training a Python ML Model Serverlessly

Problem

You have a Python model authored and want to train it, leveraging serverless compute in the cloud. You may be using one of several popular ML models, including TensorFlow, PyTorch, XGBoost, Scikit-learn, and so on.

Solution

Prepare your Python model for submission to the Cloud AI Platform training service. In this case, we will prepare a TensorFlow model, which is an open source framework to help you develop and train ML models.

1. Create a model and, if in a Jupyter Notebook, export it to a Python file. It looks something like this. For this recipe, use the GitHub repo code provided:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential


# Data Loading and engineering steps omitted
output_directory = os.environ['AIP_MODEL_DIR']


model = Sequential([
    keras.layers.Dense(
            15, activation="relu",
input_shape=(train_features.shape[-1],)
        ),
        keras.layers.Dense(10, activation="relu"),
        keras.layers.Dense(1, activation=None)
])

model.compile(loss='mae')
model.fit(train_features, train_labels, epochs=500, valida-
tion_data=(test_features, test_labels))

model.save(output_directory)
```

The model save location will be provided by the training service via an *environment* variable. This model uses a common example data set, the Boston house prices data set, where we use features about Boston neighborhoods to predict the median house value.

2. Prepare your code as a Python model. Create a *Trainer* folder, move your model to this folder, rename it *task.py*, and create an empty *__init.py__* file to make it a package.
3. Enable the Vertex AI API if you haven't already. Run this from the command line or Cloud Shell:

    ```
    gcloud services enable aiplatform.googleapis.com
    ```

4. You'll need to package your model as a gzipped tarball and stage it on GCS for the training service to access. Run the following to create the tarball, using the *setup.py* file included with the example code:

    ```
    python3 setup.py sdist --formats=gztar
    ```

5. Next, copy the tarball to your training bucket:

    ```
    gsutil cp dist/boston-housing-training-1.0.tar.gz \
      gs://BUCKET_NAME/training/
    ```

6. Now, from the Vertex AI UI, open the Training section. Click Create to start a new job.
7. Select Custom Training (Advanced), as we are using our own code. Click Continue.
8. Name the model **`boston-housing`**.
9. Choose a prebuilt container, such as TensorFlow Version 2.3.
10. Under Package Location, click Browse and navigate through your bucket to the tarball file, *boston-housing-training-1.0.tar.gz*.
11. Enter **`trainer.task`** for Python module.
12. Under Model Output Directory, enter **`gs://BUCKET_NAME/output`** and then click Continue.
13. Skip hyperparameter tuning for now; choose `n1-standard-4` for the machine type.
14. Last, configure a prebuilt serving container, choosing TensorFlow 2.3 as before. Ensure that the Model directory field is set to our output folder, and then click Start Training.
15. You can view the logs by clicking the job ID that appears. When the training job has completed, you will see your model in the Models section of the UI. The `SavedModel` output will also be staged in your GCS bucket, which you can view by running the following:

    ```
    gsutil ls gs://[BUCKET_NAME]/output
    ```

Discussion

The Vertex AI Platform training service can run any generic Python batch job but is designed for short- or long-lived ML training jobs. It also supports passing in custom command-line parameters, installing your own Python dependencies, and configuring machine types to support a wide variety of ML training requirements. Hyperparameter tuning for these jobs is easy to configure. Your model can also be automatically staged for serving, as you will see in the next recipe.

10.3 Making Serverless Predictions with a Python Model

Problem

You have a Python model authored and want to make predictions in a serverless manner in the cloud.

Solution

Use Vertex AI Serving to upload and serve a model. There are two elements of serving a model: autoscaling and batch predictions. Once serving, you can call the API to get predictions from your model, using new input data. You will first need a model trained on Vertex AI training. If you completed the previous recipe, you should have the boston-housing model listed in the Models section of the UI since we configured a prebuilt serving container.

1. Click the model name. In the Deploy & Test tab, click Deploy To Endpoint.
2. Name it **`boston-housing-endpoint`**. Choose n1-standard-2 as the machine type and leave all the other defaults. Click Deploy and wait a few minutes for the green checkmark to appear, indicating that your model is serving.
3. To test the endpoint, an *instances.json* file with some sample requests has been provided in the repo. Examine those inputs and then run the following on the command line to get some sample predictions. Replace ENDPOINT_ID and PROJECT_ID with yours.

   ```
   ENDPOINT_ID=[ENDPOINT_ID]
   PROJECT_ID=[PROJECT_ID]
   INPUT_DATA_FILE="instances.json"

   curl \
   -X POST \
   -H "Authorization: Bearer $(gcloud auth print-access-token)" \
   -H "Content-Type: application/json" \
   https://us-central1-aiplatform.googleapis.com/v1/projects/${PROJECT_ID}/
   locations/us-central1/endpoints/${ENDPOINT_ID}:predict \
   -d "@${INPUT_DATA_FILE}"
   ```

   You should see model output like the following:

   ```
   {
     "predictions": [
       [
         17.3454876
       ],
       [
         11.6165209
       ]
     ],
     "deployedModelId": "3454691922751258624"
   }
   ```

4. Remember to delete the endpoint to prevent charges.

Discussion

The Vertex AI model service allows you to host your models easily, without having to worry about managing, maintaining, or scaling underlying infrastructure. Your ML endpoint will automatically scale up and down as traffic increases or decreases. You can still configure machine type, add GPUs, and so on. Creating a model in Vertex AI like this will allow you to perform batch predictions if you want to run an inference on thousands of examples at once. You can also further configure your model to provide explanations for each feature at prediction time. That is, how much did each input contribute to the final prediction? Note that IAM permissions are not granular, so if you want to lock down different models further for different users within a project, you will want to front them with their own frontends, perhaps using Cloud Run.

10.4 Creating a Custom Notebook Environment

Problem

You need specific libraries preinstalled in your Vertex AI Notebook environment and consistency between your development environment and the production pipeline environment.

Solution

Leverage Cloud Build to add your custom libraries to a Vertex AI container, push it to a container registry, and use it to create a Vertex AI Platform Notebook.

1. On your local workstation, create a new folder or, if you'd prefer to follow along, go to the cloned repository and then to the folder for this recipe.
2. In this folder, create a file called *requirements.txt* and add the following. This will contain the Python modules you want to add to your Vertex AI Notebook, in this case Kubeflow Pipelines (kfp), to build ML training pipelines.

   ```
   kfp==0.2.5
   ```

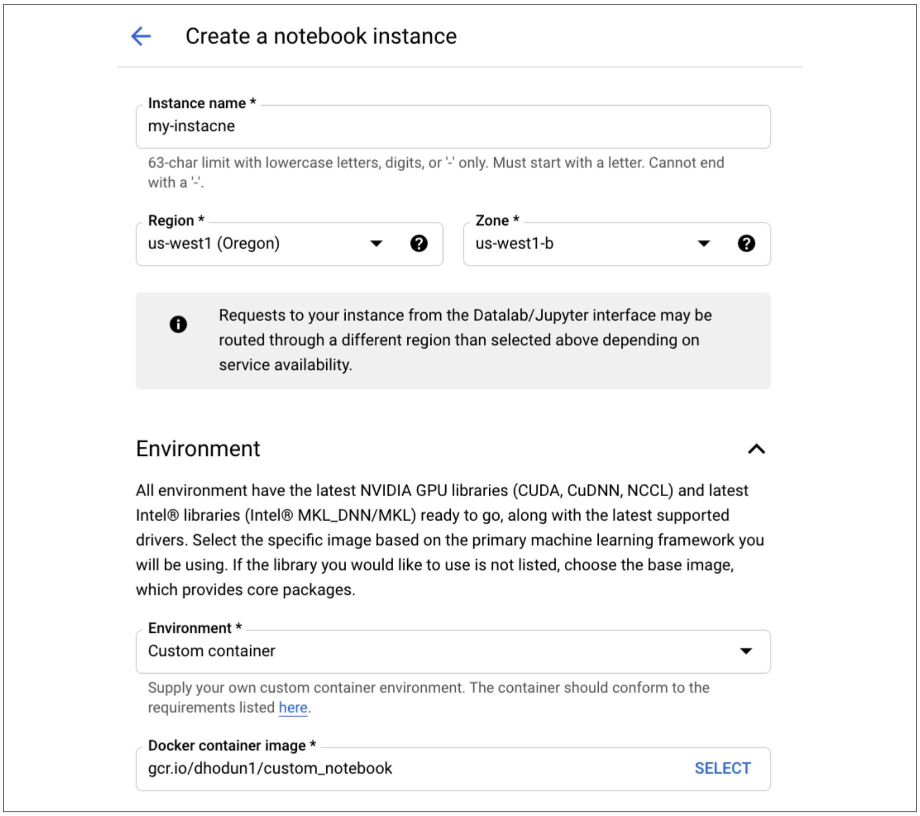
3. Create a file called *Dockerfile* in the same directory and add this code:

   ```
   FROM gcr.io/deeplearning-platform-release/base-cpu

   COPY requirements.txt .
   RUN python3 -m pip install -U -r requirements.txt
   ```

4. Run the following on the command line in this folder to submit this build to Google Cloud Build. This will create a tarball of the local directory, upload it to the Cloud Build service, build the container image, and push it to Google Container Registry. You can examine the logs that immediately start streaming from the command line or under the Cloud Build section of the Google Cloud console.

```
IMAGE_NAME=custom_notebook
TAG=latest
PROJECT_ID=$(gcloud config get-value project)

IMAGE_URI="gcr.io/${PROJECT_ID}/${IMAGE_NAME}:${TAG}"

gcloud builds submit --timeout 10m --tag ${IMAGE_URI} .
```

5. Now we can create our notebook. In the Google Cloud console, navigate to Vertex AI > Notebooks. Click + New Instance > Customize Instance.
6. Provide an instance name.
7. Under Environment, choose Custom Container.
8. Input the container location (*gcr.io/MY_PROJECT/custom_notebook:latest*).
9. Click Create. Figure 10-4 shows the completed dialog box.


Create a notebook instance

Instance name *
my-instacne
63-char limit with lowercase letters, digits, or '-' only. Must start with a letter. Cannot end with a '-'.

Region *
us-west1 (Oregon)

Zone *
us-west1-b

Requests to your instance from the Datalab/Jupyter interface may be routed through a different region than selected above depending on service availability.

Environment

All environment have the latest NVIDIA GPU libraries (CUDA, CuDNN, NCCL) and latest Intel® libraries (Intel® MKL_DNN/MKL) ready to go, along with the latest supported drivers. Select the specific image based on the primary machine learning framework you will be using. If the library you would like to use is not listed, choose the base image, which provides core packages.

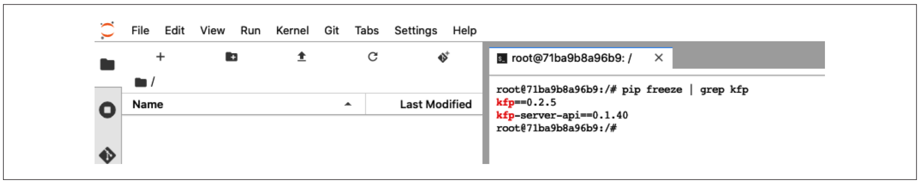
Environment *
Custom container
Supply your own custom container environment. The container should conform to the requirements listed here.

Docker container image *
gcr.io/dhodun1/custom_notebook

Figure 10-4. Creating a custom notebook

10. Once the notebook is created, the OPEN JUPYTERLAB link should be enabled. Click this.
11. Open a terminal in your notebook and run the following to see your Python dependencies installed, as shown in Figure 10-5:

    ```
    pip freeze | grep kfp
    ```



```
root@71ba9b8a96b9:/# pip freeze | grep kfp
kfp==0.2.5
kfp-server-api==0.1.40
root@71ba9b8a96b9:/#
```

Figure 10-5. The correct Python modules installed on the new notebook

12. Alternatively, you can create the notebook using the gcloud command-line tool:

    ```
    gcloud notebooks instances create my-notebook-2 --container-repository/
    gcr.io/dhodun1/custom_notebook --machine-type n1-standard-4 --location/
    us-west1-b
    ```

Discussion

Baking in custom dependencies to a Vertex AI Notebook image programmatically is often helpful when deploying a standard data science image across a team or teams, as well as when aligning dependencies in the development phase with a production environment. The base Vertex AI containers already have a large variety of data science and ML tools, which is the reason for the large container size (>1 GB), but often it will be necessary to augment or change package versions. By using the Vertex AI container as the parent image, you can retain most of the benefits of the out-of-the-box Vertex AI notebooks, such as authenticated reverse proxy, service or user account alignment, and a unified API for all notebooks, while customizing and potentially further securing the environment.

10.5 Extracting Data from BigQuery to Pandas for Model Training

Problem

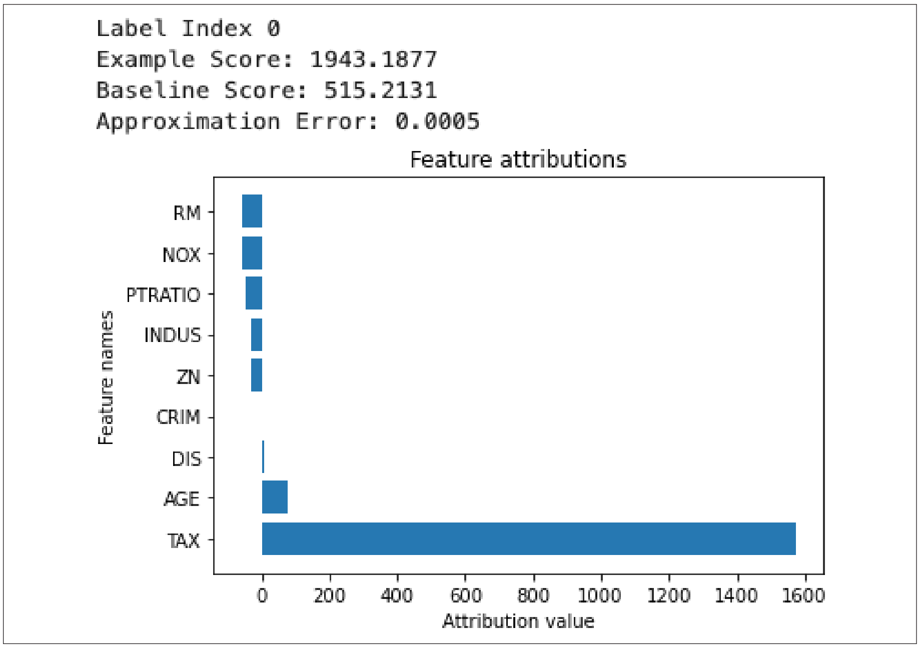
You need to build a more advanced, hand-tuned ML model in Python, using data stored in BigQuery.

Solution

Extract data from BigQuery, using both *tf.data* for large data sets and the BigQuery client to extract an in-memory data set into a pandas DataFrame.

1. In a Vertex AI notebook, clone the repository, select the recipe folder, and open the sample notebook.
2. The first option to extract data from BigQuery into memory is by using the built-in BigQuery magic and passing the name of the DataFrame to output the query, in this case *df_from_magic*. We will read from a public data set of ride data from the London Bicycle share program. Running the `DataFrame.head()` command will show you the data properly imported, as in Figure 10-6.

```
%%bigquery df_from_magic --use_bqstorage_api
SELECT * FROM
bigquery-public-data.london_bicycles.cycle_hire
WHERE EXTRACT(YEAR from start_date) = 2017
AND EXTRACT(MONTH from start_date) = 1
```



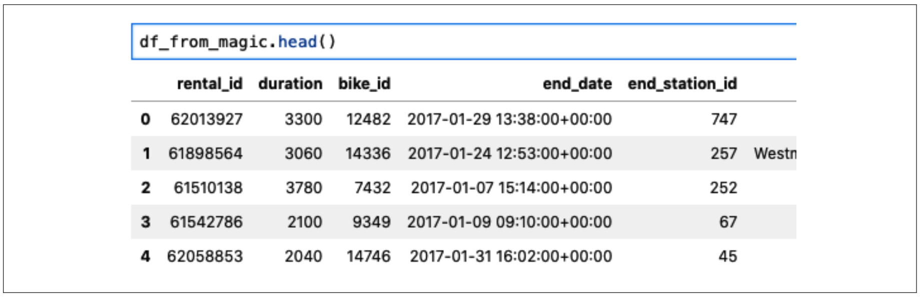
```
Label Index 0
Example Score: 1943.1877
Baseline Score: 515.2131
Approximation Error: 0.0005
```

Figure 10-6. BigQuery magic DataFrame output

Alternatively, you can load the BigQuery client directly and query that way with the following steps.

3. Run the following code in your notebook. You'll see a DataFrame similar to Figure 10-7.

```
from google.cloud import bigquery
client = bigquery.Client()

query_string = """
SELECT duration, start_station_id,
  EXTRACT(DAYOFWEEK from start_date) as day_of_week,
  EXTRACT(HOUR from start_date) as hour

FROM
bigquery-public-data.london_bicycles.cycle_hire
WHERE EXTRACT(YEAR from start_date) = 2017
AND EXTRACT(MONTH from start_date) = 1
"""
df = client.query(query_string).to_dataframe()
```



```
df_from_magic.head()
```

	rental_id	duration	bike_id	end_date	end_station_id	
0	62013927	3300	12482	2017-01-29 13:38:00+00:00	747	
1	61898564	3060	14336	2017-01-24 12:53:00+00:00	257	Westn
2	61510138	3780	7432	2017-01-07 15:14:00+00:00	252	
3	61542786	2100	9349	2017-01-09 09:10:00+00:00	67	
4	62058853	2040	14746	2017-01-31 16:02:00+00:00	45	

Figure 10-7. BigQuery client DataFrame output

Next, build a `tf.data` object from the DataFrame to then feed into your model.

4. Run the following code:

```
import tensorflow as tf

target = df.pop('duration')
dataset = tf.data.Dataset.from_tensor_slices((df.values, target.values))

for feat, targ in dataset.take(5):
  print ('Features: {}, Target: {}'.format(feat, targ))

train_dataset = dataset.shuffle(len(df)).batch(64).prefetch(1)
# 1=AUTOTUNE
```

5. Build and train your model with the `tf.data` set:

```
from tensorflow import keras

# Simple model shown for simplicity and using the tf.data API

model = keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_absolute_error')
    model.fit(train_dataset, epochs=2)
```

Discussion

If you are building a Python-based model outside of BigQuery or AutoML and the data set can fit in memory, extracting a BigQuery training set to a pandas DataFrame is an excellent way to feed your model. You can easily add large quantities of memory to your Vertex AI Notebook VM for a few minutes if needed for a one-time training run. In the case of TensorFlow, `tf.data.Dataset.from_tensor_slices()` will accept dataframes. There are other methods for other frameworks. If your data set is larger than what can be stored in memory, it is recommended to export the training data to multiple files on GCS, either in CSV format or in the highly optimized TFRecord format.

10.6 Training a Model in SQL with BQML

Problem

You want to train an ML model easily, without Python code and using data already in BigQuery.

Solution

BQML allows you to train simple and sophisticated models with SQL all in the BigQuery service. You can easily create features and preprocess data as well.

In this example, we will use a public data set containing New York City taxi ride records. We want to predict the eventual fare of a taxi, given some input features we will know at the start of the ride.

1. Examine the data set by running the following in the BigQuery UI:

```
SELECT *
FROM
  `bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2018`
WHERE
  EXTRACT(DATE FROM pickup_datetime) = "2018-01-28"
```

2. Run the following SQL to define and train your model:

```
CREATE OR REPLACE MODEL
  mydataset.taxi_model OPTIONS(model_type='linear_reg',
    input_label_cols=['fare_amount']) AS
SELECT
  fare_amount,
  CAST(pickup_location_id AS string) AS pickup_location,
  CAST(dropoff_location_id AS string) AS dropoff_location,
  CAST(EXTRACT(HOUR FROM pickup_datetime) AS string) AS hour,
  CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS string) AS
day_of_week,
FROM
  `bigquery-public-data`.new_york_taxi_trips.tlc_yellow_trips_2018
WHERE
    EXTRACT(DATE FROM pickup_datetime) < "2018-07-01"
```

We are choosing a linear regression model and giving features that are known at the beginning of the ride.

> We cast features like `pickup_location_id` to `STRING` so they are treated as categorical features rather than numeric features. We'll use data from the first half of 2018.

Once the model is finished training, you can examine the model training metrics, either by opening the model from the browser on the left or by running the following SQL. In my case, the mean absolute error was 5.9, not too bad for a first-pass model.

```
SELECT * FROM ML.EVALUATE(MODEL mydataset.taxi_model)
```

3. Last, let's make some predictions, using our new model and the second half of the year by running this SQL:

```
SELECT *
    FROM
      ML.PREDICT(MODEL `mydataset.taxi_model`,
        (
        SELECT
          fare_amount,
          CAST(pickup_location_id AS string) AS pickup_location,
          CAST(dropoff_location_id AS string) AS dropoff_location,
          CAST(EXTRACT(HOUR FROM pickup_datetime) AS string) AS hour,
          CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS string) AS
day_of_week,
        FROM
          `bigquery-public-
data`.new_york_taxi_trips.tlc_yellow_trips_2018
        WHERE
          EXTRACT(DATE FROM pickup_datetime) = "2018-07-01"
        LIMIT 20 ))
```

Using this method, it is very easy to make a large batch of predictions and store them in BigQuery for future use or lookup.

Discussion

BQML is a powerful and surprisingly easy-to-use tool for performing large-scale machine learning. Many model architectures are supported, such as neural networks, gradient-boosted trees, time-series approaches, and matrix factorization recommender algorithms. You can also perform more advanced feature engineering in SQL, such as feature crossing. You may also want to examine the `TRANSFORM()` clause, which captures transformations you made to create your features and applies them automatically to incoming data on prediction. (You'll notice we had to copy the transformation logic from our training SQL to our inference SQL, since we didn't use `TRANSFORM()`.) Even if you're a seasoned Python ML engineer, BQML is certainly worth a look—you'll be surprised at how quickly you can get large-scale, high-performing models up and running.

CHAPTER 11

Google Cloud Security and Access

Google Cloud provides a robust set of services to secure your Google Cloud organization and projects. These services are continually being updated, so we recommend that you visit the Google Cloud documentation as well as review Google Cloud's best practices website.

In this chapter, you will learn how to create a service account to allow applications to access Google Cloud resources securely, how to implement authentication for applications running on Google Kubernetes Engine (GKE), how to run asset reports, and how to build a deny-and-allow list for your applications.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.

11.1 Creating a Service Account

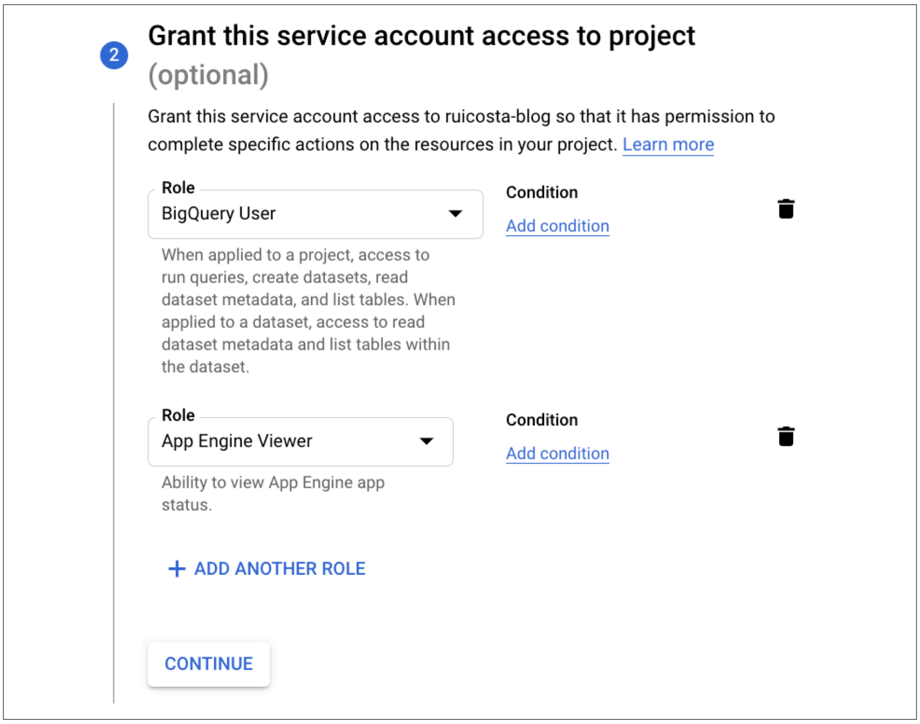
Problem

You need to authorize your application to access resources securely on Google Cloud.

Solution

Using service accounts, you can make authorized API calls to Google Cloud and restrict the service account permissions to only what is required by the application. In this recipe, you will learn how to create a service account through the Google Cloud Console.

1. In the Cloud Console, open IAM & Admin and choose Service Accounts.
2. Click Create Service Account.
3. Enter a service account name.
4. Click Create And Continue to the next step.
5. Choose one or more Identity and Access Management (IAM) roles to grant to the service account on the project, as shown in Figure 11-1.



Grant this service account access to project (optional)

Grant this service account access to ruicosta-blog so that it has permission to complete specific actions on the resources in your project. Learn more

Role: BigQuery User

When applied to a project, access to run queries, create datasets, read dataset metadata, and list tables. When applied to a dataset, access to read dataset metadata and list tables within the dataset.

Condition: Add condition

Role: App Engine Viewer

Ability to view App Engine app status.

Condition: Add condition

Figure 11-1. Grant a role to a service account

6. When you finish adding roles, click Continue.
7. Click Done to finish creating the service account.

Discussion

Applications use a service account to access resources securely in Google Cloud. Service accounts do not have a password; they are associated with private and public RSA key pairs for authentication. You restrict access to a service account, only providing it access to the resources required. Minimizing the access of the service account will limit the exposure or risk if the service account is compromised. It is always best practice to use service accounts when you need applications running on servers to communicate with Google Cloud resources. It is not advisable to use service accounts on user applications, because this would authenticate users and use a RESTful API running within your project.

11.2 Creating Custom Roles to Access a Cloud Storage Bucket

Problem

You want a newly created service account to list storage buckets for your application.

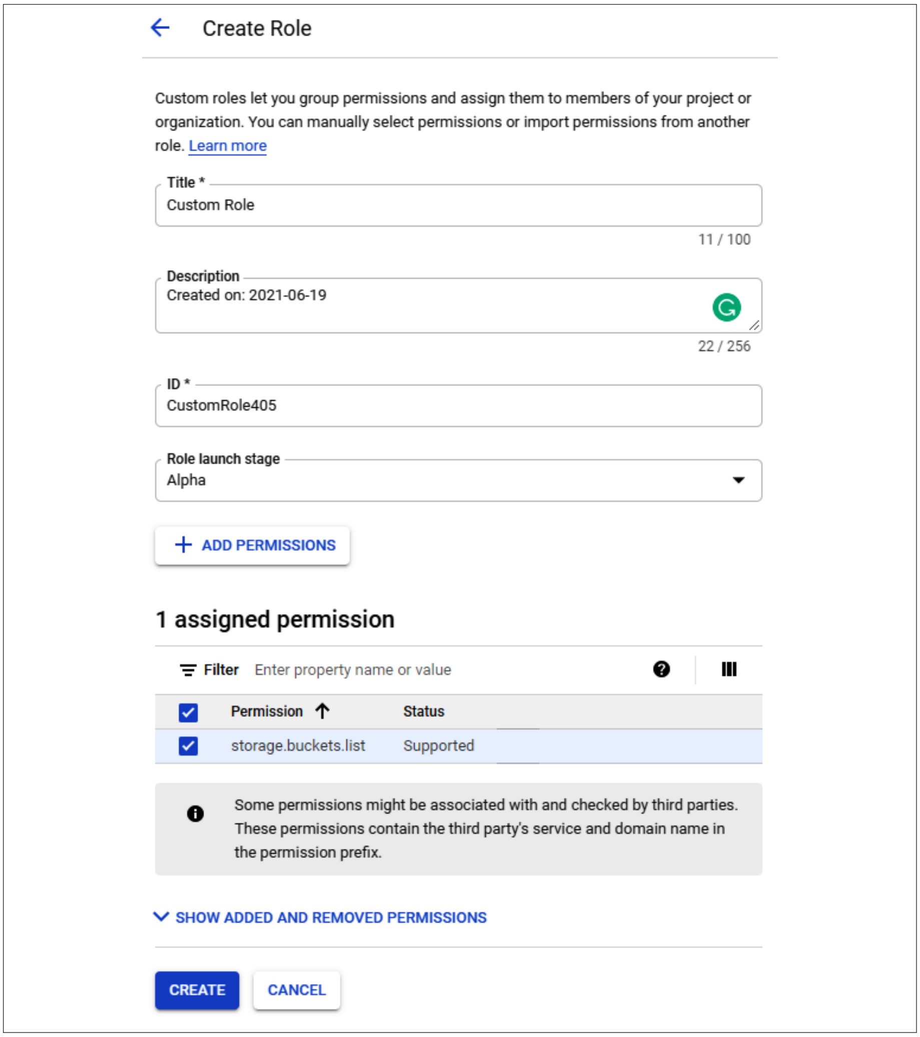
Solution

Create custom roles to allow you to assign `storage.buckets.list` to a service account to be able to list buckets in your application.

1. In the Cloud Console, from IAM & Roles, open the Roles page.
2. Click Create Role.
3. Enter a Title, Description, ID, and Role launch stage for the custom role.

> The role name cannot be changed after the role is created.

4. Click Add Permissions.
5. In the Enter property name or value input field, enter `storage.buckets.list` permissions, and select the role as shown in Figure 11-2.


Create Role

Custom roles let you group permissions and assign them to members of your project or organization. You can manually select permissions or import permissions from another role. Learn more

Title *
Custom Role
11 / 100

Description
Created on: 2021-06-19
22 / 256

ID *
CustomRole405

Role launch stage
Alpha

1 assigned permission

✓	Permission	Status
✓	storage.buckets.list	Supported

Some permissions might be associated with and checked by third parties. These permissions contain the third party's service and domain name in the permission prefix.

Figure 11-2. Creating a custom IAM role

6. Click Create.

Discussion

You cannot list storage buckets with the Storage Object Viewer role. It only provides you with the following permissions:

- *resourcemanager.projects.get*
- *resourcemanager.projects.list*
- *storage.objects.get*
- *storage.objects.list*

Due to working with least privilege, we can create a custom role to avoid giving service accounts elevated privileges. Create a custom role with the `storage.buckets.list` permission and then assign it to your service account. You have thus prevented limited permissions but still provided your application with the ability to list storage buckets.

11.3 Authenticating an Application Running on Kubernetes Engine

Problem

You need to enable authentication for an application that is running on Google Cloud Kubernetes Engine (GKE).

Solution

Using Identity-Aware Proxy, secure your application so that it requires users to authenticate before they can access the application running on GKE.

Prerequisites

You will need to have a registered domain name for this recipe.

1. Create a GKE cluster by running the following command:

   ```
   gcloud container clusters create hello-cluster
   ```

   When creating a GKE cluster, using the `gcloud` command, the configuration context of the cluster is added to `kubeconfig`, allowing you to use the `kubectl` command. The *kubeconfig* file organizes information about clusters, users, and namespaces.

2. Install a sample application to GKE by creating the following two declarative YAML files:

   ```
   deploy.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: web
     namespace: default
   spec:
     selector:
       matchLabels:
         run: web
     template:
       metadata:
         labels:
           run: web
       spec:
         containers:
         - image: gcr.io/google-samples/hello-app:2.0
           imagePullPolicy: IfNotPresent
           name: web
           ports:
           - containerPort: 8080
             protocol: TCP

   service.yaml
   apiVersion: v1
   kind: Service
   metadata:
     name: web
     namespace: default
   spec:
     ports:
     - port: 8080
       protocol: TCP
       targetPort: 8080
     selector:
       run: web
     type: NodePort
   ```

3. Apply the configuration to your cluster by running the following command:

   ```
   kubectl apply -f deploy.yaml
   kubectl apply -f service.yaml
   ```

4. Reserve a global IP address to be used for the Google Cloud Load Balancer (GCLB):

   ```
   gcloud compute addresses create hello-static-ip --global
   ```

5. Create an *ingress.yaml* declarative file. This declarative file declares a GCLB to be created, and it will route the traffic to the Global IP address reserved in step 4:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
  annotations:
    kubernetes.io/ingress.global-static-ip-name: "hello-static-ip"
spec:
  backend:
    serviceName: web
    servicePort: 8080
```

6. Apply the configuration to your cluster by running the following command:

```
kubectl apply -f ingress.yaml
```

7. To find the reserved static IP addresses, run the following command:

```
gcloud compute addresses describe hello-static-ip --global
```

8. Update or create a DNS A record to point to the static IP address you reserved in step 4 and listed in step 7.
9. Create a *certificate.yaml* declarative file, which will create a new managed certificate, and replace `example.com` with your DNS A record:

```
apiVersion: networking.gke.io/v1beta1
kind: ManagedCertificate
metadata:
  name: iap
spec:
  domains:
    - example.com
```

10. Update the *ingress.yaml* declarative file to include:

```
networking.gke.io/managed-certificates: "iap"
```

The *update ingress.yaml* file should look like this:

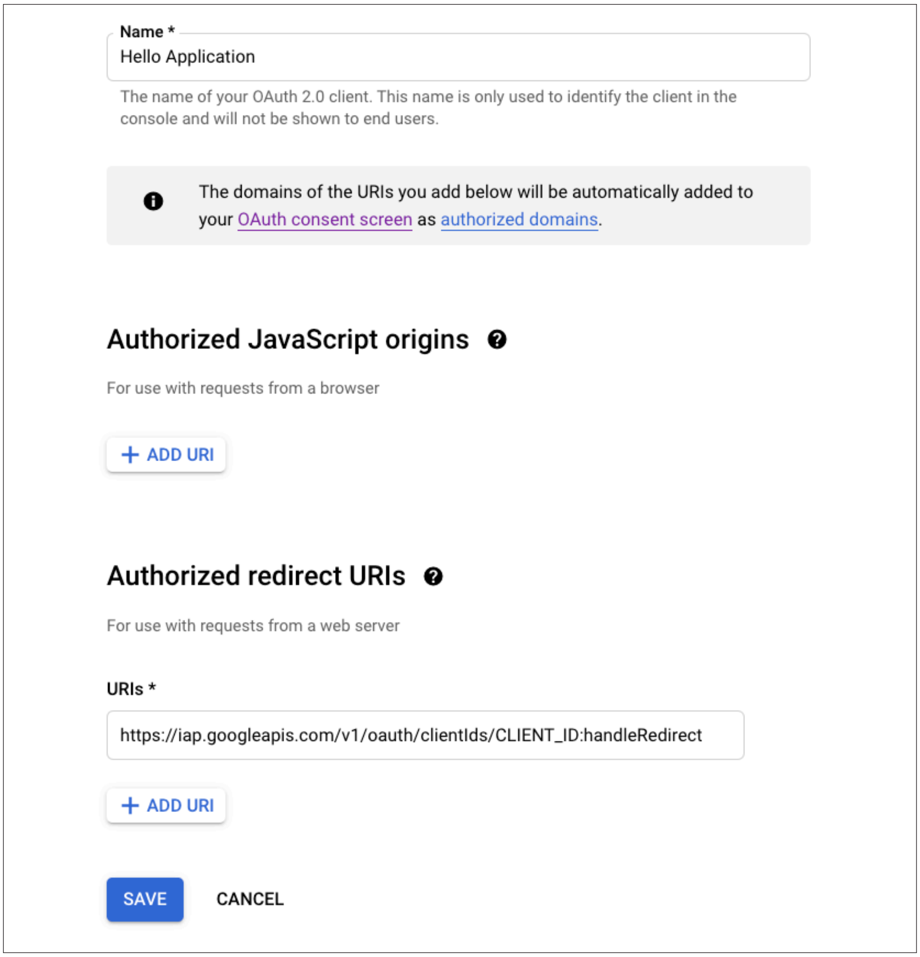
```
    apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
  annotations:
    kubernetes.io/ingress.global-static-ip-name: "hello-static-ip"
    networking.gke.io/managed-certificates: "iap"
spec:
  backend:
    serviceName: web
    servicePort: 8080
```

11. Apply the configuration to your cluster by running the following command:

```
kubectl apply -f certificate.yaml
kubectl apply -f ingress.yaml
```

It will take a few minutes for the certificate to be provisioned.

12. To enable Identity-Aware Proxy, configure the OAuth consent screen and create OAuth credentials:
    a. In the Google Cloud Console, open APIs & Services and select the OAuth consent screen.
    b. Under Support email, select the email address you want to display as a public contact.
    c. When someone tries to access your application and has some type of error with authentication, this email address will be presented for support.
    d. Enter the application name you want to display.
    e. Click Save.
    f. In the Google Cloud Console, open APIs & Services and choose Credentials.
    g. From the Create Credentials drop-down list, select OAuth client ID.
    h. Under Application Type, select Web Application.
    i. Add a name for your OAuth client ID.
    j. Click Create. Your OAuth client ID and client secret are generated and displayed in the OAuth client window.
    k. Copy the client ID to the clipboard. Click OK.
    l. Select the newly created OAuth Client ID.
    m. Add *https://iap.googleapis.com/v1/oauth/clientIds/CLIENT_ID:handleRedirect* to the Authorized Redirect URIs, as shown in Figure 11-3 and replace the CLIENT_ID with the client ID copied previously


Name *

Hello Application

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.

Authorized JavaScript origins

For use with requests from a browser

Authorized redirect URIs

For use with requests from a web server

URIs *

https://iap.googleapis.com/v1/oauth/clientIds/CLIENT_ID:handleRedirect

Figure 11-3. Adding a URI to the OAuth ID

13. Near the top of the page, click Download JSON. You'll need the values within the JSON file in step 15. You can open the JSON file with your favorite editor.
14. Set up Identity-Aware Proxy access:
    a. In the Google Cloud Console, open Security and choose Identity-Aware Proxy.
    b. Select the checkbox next to the resource you want to add members to.
    c. On the right-side panel, click Add Member.
    d. In the Add Members dialog box that appears, enter the email addresses of who should have the IAP-secured Web App User role for the project.

   e. Select Cloud IAP > IAP-Secured Web App User from the Roles drop-down list.

   f. Click Save.

   g. Enable IAP on for your service by toggling the on/off switch.

15. Apply a secret to GKE by running the following command:

    ```
    kubectl create secret generic iap-oauth \
    --from-literal=client_id=YOUR_CLIENT_ID \
    --from-literal=client_secret=YOUR_CLIENT_SECRET
    ```

    Replace *YOUR_CLIENT_ID* and *YOUR_CLIENT_SECRET* with the respective values. Use the JSON credentials you downloaded in step 13 to get your values.

16. Create the following declarative *backend-config.yaml* file:

    ```
    apiVersion: cloud.google.com/v1beta1
    kind: BackendConfig
    metadata:
      name: config-default
      namespace: default
    spec:
      iap:
        enabled: true
        oauthclientCredentials:
          secretName: iap-oauth
    ```

17. Apply the configuration to your cluster by running the following command:

    ```
    kubectl apply -f backend-config.yaml
    ```

18. Now visit the URL of the DNS A record you created; you should be prompted to authenticate.

19. Sign in with the user you granted the IAP-Secured Web App User role.

20. You should now be authenticated to the Hello application running on GKE.

Discussion

Identity-Aware Proxy (IAP) allows you to secure access to your applications running on GKE, Compute Engine, and App Engine. IAP provides a mechanism to secure your applications within minimal code requirements on your application. It is still recommended to check the user's bearer token within your application; however, all authentication is done by IAP. IAP also supports external identities such as Facebook, GitHub, Microsoft, Phone Number, and so on.

11.4 Retrieving the Authenticated User's Identity

Problem

You have enabled Identity-Aware Proxy (IAP) and want to retrieve the authenticated user's email address.

Solution

Using the HTTP headers passed by Identity-Aware Proxy (IAP), you can access the user's email address and their user ID.

1. Follow the steps in Recipe 11.3 to deploy the sample application and enable Identity-Aware Proxy. In step 2, replace the container image path with:

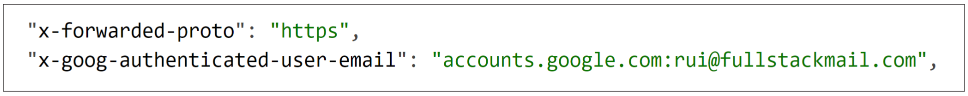
   ```
   - image: us.gcr.io/ruicosta-blog/hello-java-iap:v5
   ```

2. The sample container image is a Java Spring Boot application. It reads the HTTP headers, which include the following attributes from IAP:

Header name	Description
X-Goog-Authenticated-User-Email	The user's email address
X-Goog-Authenticated-User-Id	A persistent, unique identifier for the user

3. Visit the hostname of your deployed application from step 1 and sign in with the user you granted access to.
4. Once signed in, click the Generate Headers button.

You will now be presented with the request headers, as shown in Figure 11-4. You will notice you can now see who the user is who signed in.



```
"x-forwarded-proto": "https",
"x-goog-authenticated-user-email": "accounts.google.com:rui@fullstackmail.com",
```

Figure 11-4. Request headers

Discussion

Identity-Aware Proxy (IAP) adds headers to the user request, allowing you to know easily who the authenticated user is. By having this information, you can provide a customized experience for the user as well as provide role-based access, depending on who the authenticated user is.

To read the headers in Java, the following code snippet just does a `forEach` through all the headers seen by Spring Boot:

```
headers.forEach((key,value) ->{
System.out.printf("Header Name: "+key+"
          Header Value: "+value);
});
```

11.5 Authenticating a Java Application Using a Service Account

Problem

You need your Java application to access Google Cloud Storage buckets securely and with least privilege access roles.

Solution

Use IntelliJ and a Google Cloud service account to run a sample application that will securely list the buckets that you have in your Google Cloud project.

1. On your local workstation, go to the working directory for this demo from the cloned repository.

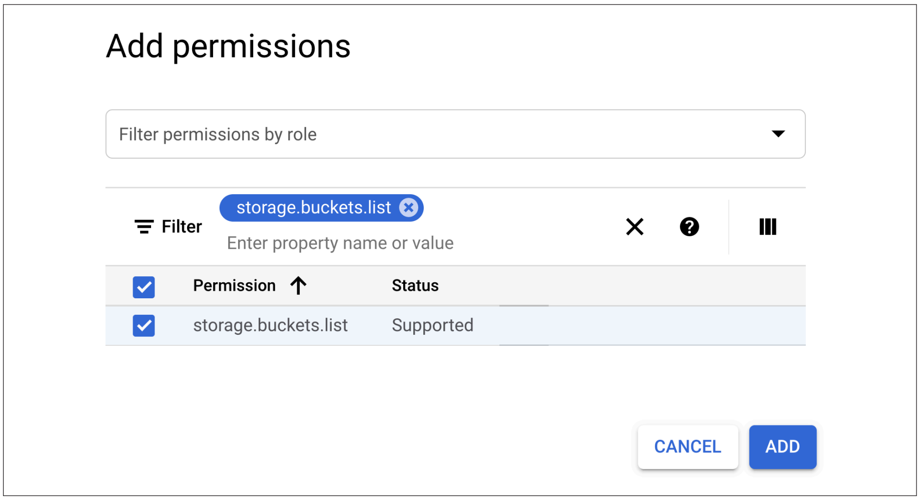
   ```
   cd 11-security-access/11-04-service-account
   ```

2. Create a service account, as described in Recipe 11.1, and assign the Storage Object Viewer role.
3. Select the newly created service account in the Cloud Console and click the Keys tab.
4. Select Create New Key.
5. Choose JSON and click Create. This will download the credentials file to your local workstation. Rename the file to **service-account.json** and copy the file to the following folder: *11-security-access/11-04-service-account/src*.
6. To test the application, in your IntelliJ IDE, run `SimpleApp.main()`.
7. The application should fail, and you should receive a message similar to the following:

   ```
   storage-view@ruicosta-blog.iam.gserviceaccount.com does not have stor-
   age.buckets.list access to the Google Cloud project.
   ```

8. This is failing because the Storage Object Viewer does not have the `storage.buckets.list` permission.
9. In the Google Cloud Console, go to the navigation menu and choose IAM & Admin > IAM.

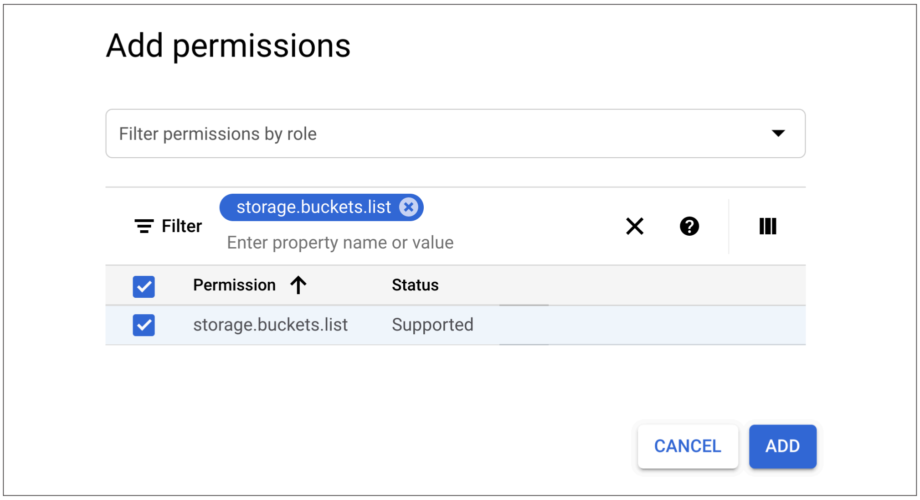
10. Edit the service account you created in step 2.
11. Click Add Another Role.
12. Select Manage Roles from the Select a Role dropdown menu.
13. Select Create Role. Enter a Title, Description, ID and Role launch Stage.
14. Select Add Permissions.
15. Add the `storage.buckets.list` permission and click ADD, as shown in Figure 11-5.



	Permission	Status
✓	storage.buckets.list	Supported

Figure 11-5. Add permissions dialog box

16. Click Create.
17. In the Google Cloud Console, from the navigation menu, click IAM & Admin and choose IAM.
18. Edit the service account you created in step 2.
19. Click ADD ANOTHER ROLE.
20. Choose the custom role you created as shown in Figure 11-6.


Add permissions

Filter permissions by role

Filter: storage.buckets.list

Enter property name or value

Permission	Status
✓ storage.buckets.list	Supported

Figure 11-6. Adding a custom role

21. Click Save.
22. To test the application, in your IntelliJ IDE, run `SimpleApp.main()`.
23. You should now see the buckets in your project listed in IntelliJ.

Discussion

Authenticating with service accounts should be used only for server-to-server requirements. You should avoid including service account credential files in user applications. If you need to allow a user to access Google Cloud resources, the recommended approach would be to have the user access a RESTFul API that has the required privileges, have the RESTFul API authenticate or authorize the user with an identity management system such as Firebase, and, once authenticated/authorized, allow the RESTFul API to make the request on behalf of the user.

11.6 Building Reports Using the Cloud Asset API

Problem

You need to build a report that lists all the resources that are being used in your Google Cloud project.

Solution

Using the Cloud Asset API, you will export inventory data from your Google Cloud project to a BigQuery table, allowing you to build reports that include a list of resources being used in your project.

1. Enable the Cloud Asset API by running the following command:

   ```
   gcloud services enable cloudasset.googleapis.com
   ```

2. Run the following command to list all the resources in your project:

   ```
   gcloud asset search-all-resources
   ```

3. To search or list only Cloud Run resources, run the following command:

   ```
   gcloud asset search-all-resources \
     --asset-types=”run.googleapis.com/Service”
   ```

4. To display only the fields you need, and to format them in a more readable format, run the following command:

   ```
    gcloud asset search-all-resources \
       --asset-types="run.googleapis.com/Service" \
       --page-size=50 \
       --format="table(displayName, assetType, location)"
   ```

5. To produce reports, you can export your asset snapshot to BigQuery and run the following command to export your assets to the defined BigQuery table:

   ```
   gcloud asset export \
       --content-type CONTENT_TYPE \
       --project 'PROJECT_ID' \
       --snapshot-time 'SNAPSHOT_TIME' \
       --bigquery-table 'BIGQUERY_TABLE' \
       --output-bigquery-force
   ```

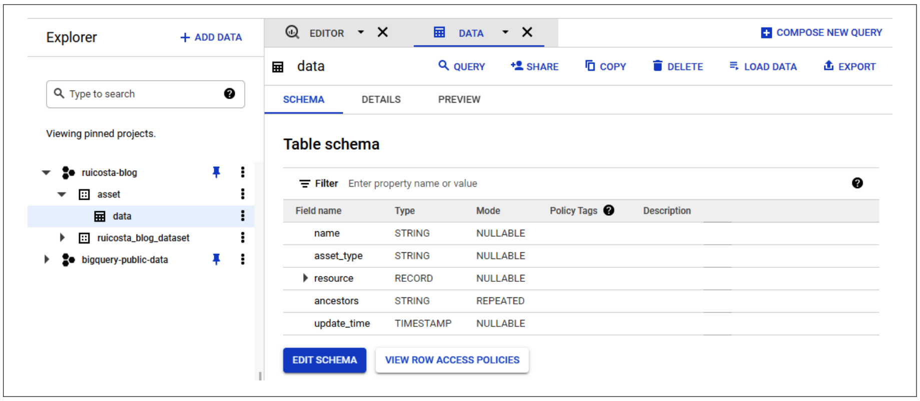
   Here is a sample command:

   ```
   gcloud asset export --content-type resource --project ruicosta-blog --
   bigquery-table
   "projects/ruicosta-blog/datasets/asset/tables/data" --output-bigquery-
   force
   ```

   A summary of the arguments used in the gcloud commands follows:

   - CONTENT_TYPE is the asset content type.
   - PROJECT_ID is the ID of the project whose metadata is being exported.
   - SNAPSHOT_TIME (Optional) is the time at which you want to take a snapshot of your assets.
   - BIGQUERY_TABLE is the table to which you're exporting your metadata, formatted *projects/PROJECT_ID/datasets/DATASET_ID/tables/TABLE_NAME*.

6. To query the output of the asset inventory to BigQuery, in the Google Cloud Console, from the navigation menu and choose Big Data > BigQuery > SQL workspace.
7. Select your data set and table and click Compose New Query, as shown in Figure 11-7.



Field name	Type	Mode	Policy Tags	Description
name	STRING	NULLABLE		
asset_type	STRING	NULLABLE		
resource	RECORD	NULLABLE		
ancestors	STRING	REPEATED		
update_time	TIMESTAMP	NULLABLE		

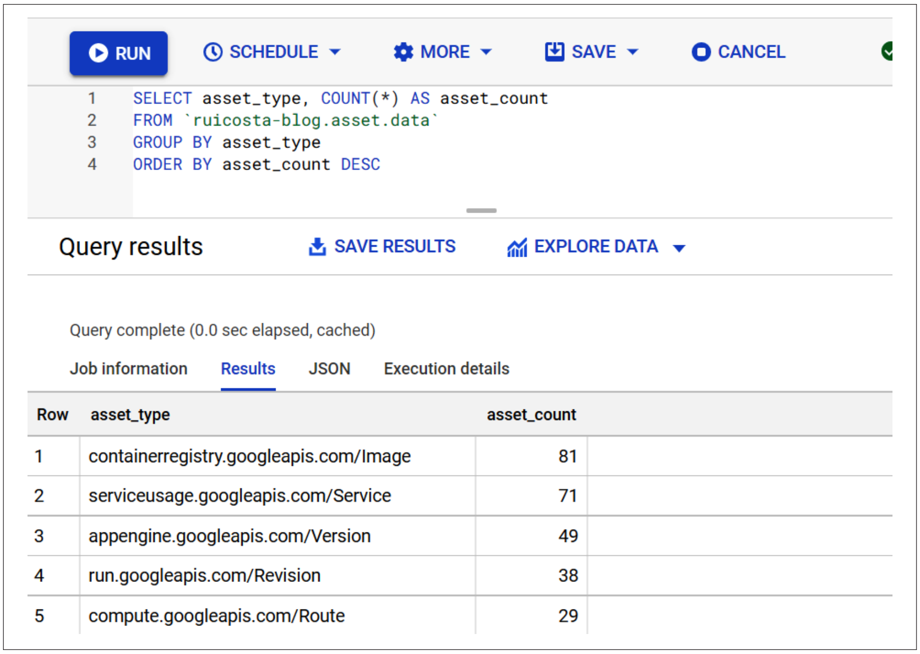
Figure 11-7. Compose a new query

8. Enter the following SQL statement in your query editor and change *PROJECT_ID.DATASET_ID.TABLE_NAME* to your respective project, data set, and table:

```
SELECT asset_type, COUNT(*) AS asset_count
FROM `PROJECT_ID.DATASET_ID.TABLE_NAME`
GROUP BY asset_type
ORDER BY asset_count DESC
```

9. Click RUN.

   You should now be able to see your results in the Query Results panel, as shown in Figure 11-8.



```
SELECT asset_type, COUNT(*) AS asset_count
FROM `ruicosta-blog.asset.data`
GROUP BY asset_type
ORDER BY asset_count DESC
```

Query complete (0.0 sec elapsed, cached)

Row	asset_type	asset_count
1	containerregistry.googleapis.com/Image	81
2	serviceusage.googleapis.com/Service	71
3	appengine.googleapis.com/Version	49
4	run.googleapis.com/Revision	38
5	compute.googleapis.com/Route	29

Figure 11-8. BigQuery results

Discussion

A key to managing a secure Google Cloud project is knowing what is being used in the project. By using the Cloud Asset Inventory API, you can build reports that provide historical data on the resources being used in your project. This allows you to control the costs as well as understand what is being used. Say you want to list compute instances that are not active so you can delete these resources. To list the compute instances that are not active, run the following gcloud command:

```
gcloud asset search-all-resources \
--query='NOT state:active' \
--scope=organizations/123456 \
--asset-types='compute.googleapis.com/Instance' \
--page-size=50 \
--format='table(name, assetType, state)'
```

11.7 Allowing a List of IP Addresses to Access Your Application

Problem

You need a method to create an allow list of IP addresses that can access your application running on Google Kubernetes Engine.

Solution

Use security policies to create a deny and allow list to your application.

1. Follow the steps in Recipe 11.3 to deploy a sample application with IAP enabled.
2. Create the Google Cloud Armor security policy to deny IP addresses:

   ```
   gcloud compute security-policies create \
       deny-clients-policy \
       --description "policy for external users"
   ```

3. Update the default rules to the security policies to deny traffic:

   ```
   gcloud compute security-policies rules update 2147483647 \
       --security-policy deny-clients-policy \
       --action "deny-404"
   ```

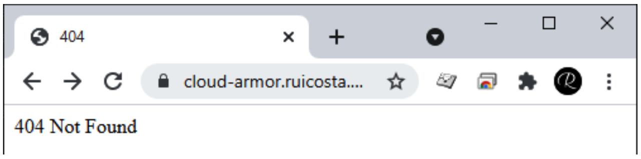
4. Run the following command to list your backend services. You will need the backend service name in step 5:

   ```
   gcloud compute backend-services list
   ```

5. Attach the security policies to the backend services:

   ```
   gcloud compute backend-services \
       update config-default \
       --security-policy deny-clients-policy
   ```

6. Choose Global when presented with Google Cloud Regions.
7. Visit your sample application with the hostname you set; you should now see the message shown in Figure 11-9.



404 Not Found

Figure 11-9. Denied access

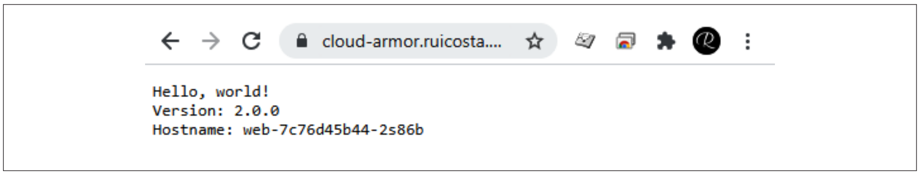
8. To allow a workstation or network to access the site, replace *MY_IP_ADDRESS* with your public IP address on your local workstation:

```
gcloud compute security-policies rules create 1000 \
    --security-policy internal-users-policy \
    --description "allow traffic personal workstation" \
    --src-ip-ranges "MY_IP_ADDRESS" \
    --action "allow"
```

9. Attach the security policies to the backend services:

```
gcloud compute backend-services \
    update config-default \
    --security-policy deny-clients-policy
```

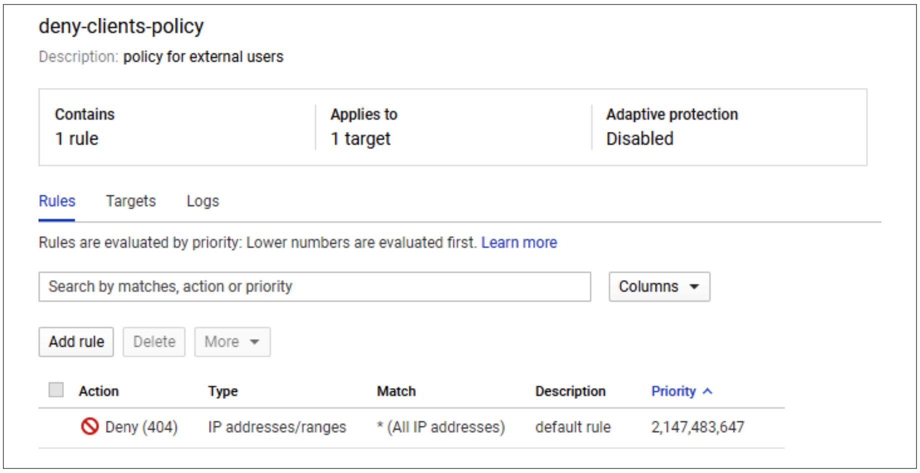
10. Choose Global when presented with Google Cloud Regions. It will take a few minutes for the policy to be enforced. After it has been applied, you should now see your application, as shown in Figure 11-10.



```
Hello, world!
Version: 2.0.0
Hostname: web-7c76d45b44-2s86b
```

Figure 11-10. Allowed access

11. You can also view your policies in the Google Cloud Console. From the navigation menu, select NETWORKING > Network Security. You should see the policy and the associated details you created in prior steps, as shown in Figure 11-11.



deny-clients-policy

Description: policy for external users

Contains	Applies to	Adaptive protection
1 rule	1 target	Disabled

Rules are evaluated by priority: Lower numbers are evaluated first. Learn more

Action	Type	Match	Description	Priority
Deny (404)	IP addresses/ranges	* (All IP addresses)	default rule	2,147,483,647

Figure 11-11. Security policy

Discussion

Google Cloud Armor security policies protect your application by providing Layer 7 filtering. Each security policy is made up of a set of rules that filter traffic based on conditions such as an incoming request's IP address, IP range, region code, or request headers. In this recipe, you used the incoming request's IP address to restrict access to your application, based on your IP address, and blocked all other traffic. You secured your application with IAP and then enforced an allow and deny list to further secure your application running on GKE.

CHAPTER 12
Google Cloud Networking

Google Cloud networking provides a robust set of services to manage networking functionality in the cloud, from securing resources to providing global load balancing to your applications. This chapter covers concepts that users require to get started with Google Cloud networking, including securing your virtual machines, automating deployments of networking resources, and protecting your projects from data exfiltration. Google Cloud networking can be a book in itself; here we provide you with concepts that are often asked by new Google Cloud users.

All code samples for this chapter are in this book's GitHub repository. You can follow along and copy the code for each recipe by going to the folder with that recipe's number.

You will need to make sure you have met the prerequisites before running through the recipes:

1. Signed up for a Google Cloud account, as described in Chapter 1.
2. Created a Google Cloud project, as described in Chapter 1.
3. Installed and configured gcloud, as described in Chapter 1.

12.1 Creating a Custom Mode VPC Network

Problem

You want to create a virtual version of a physical network in Google Cloud, with custom-defined subnet ranges.

Solution

Create a virtual private cloud (VPC) network in Google Cloud that will allow you to create a virtual network as a global resource and then define the required subnets.

1. In the Google Cloud Console, go to NETWORKS > VPC networking > VPC networks.
2. Click Create VPC Network.
3. Enter a name for the VPC network.
4. Choose Custom for the Subnet Creation mode.
5. In the New Subnet section, specify the following configuration parameters for a subnet:
   a. Provide a name for the subnet.
   b. Select a region.
   c. Enter an IP address range.
   d. Choose whether to enable Private Google Access for the subnet when you create it.
   e. Choose whether to enable VPC flow logs.

> Private Google Access allows virtual machines instances that do not have a public IP address to access most of Google APIs, please review https://cloud.google.com/vpc/docs/private-google-access for what Google APIs are not supported.

6. Click Done.
7. Leave all other settings set to default.
8. Click Create.

Discussion

You have successfully created a custom VPC network with your defined subnets. VPC networks with their associated routes and firewall rules are global resources. They are not attached or associated with a region or zone.

12.2 Creating a Static External IP Address

Problem

You have a web server running on a Google Cloud Compute Engine instance. You need to create a DNS A record so users can access the web server via its fully qualified domain name. For this you want your IP address to be static.

Solution

Reserve an external static IP address, using the gcloud command tool. Run the following command in your terminal or in the Google Cloud Shell, replacing the `ADDRESS_NAME` with a name you want for the reserved IP address:

```
gcloud compute addresses create ADDRESS_NAME \
    --global \
    --ip-version [IPV4 | IPV6]
```

You have now reserved an external IP address with Google Cloud and can use this IP address for your web server running on Google Compute Engine.

Discussion

In step 1, you ran the `gcloud compute addresses create` command, which reserved a publicly addressable IP address that allows all users to access your web server.

12.3 Create a Firewall Rule

Problem

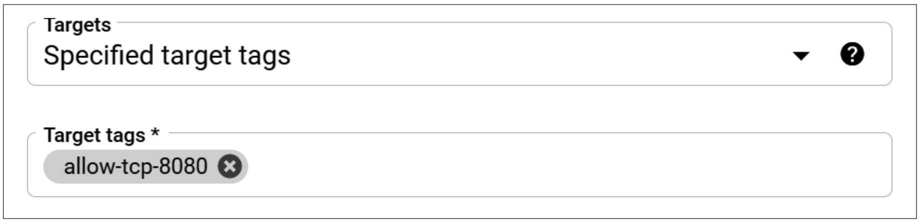
You have an application running on a virtual machine, and you need to allow everyone to access it publicly running on TCP port 8080.

Solution

Using the Google Cloud Console, create an ingress firewall rule allowing all IP addresses to access your application running on a virtual machine using TCP port 8080. You will use 0.0.0.0/0 as the source IP ranges, which means any IP address can access your application.

1. In the Google Cloud Console, navigate to Networking > VPC Network > Firewall.
2. Click Create Firewall Rule.
3. Enter a name for the firewall rule.

4. Optionally, you can enable firewall rules logging.
5. Specify the network for the firewall rule.
6. Specify the priority of the rule. The lower the number, the higher the priority.
7. For the direction of traffic, choose Ingress.
8. For Action On Match, choose Allow.
9. To specify the targets of the rule, select Specified Target Tags To Apply This Rule To Only Specific Instances and then type the tags to which the rule should apply in the Specified Target Tags field, as shown in Figure 12-1.

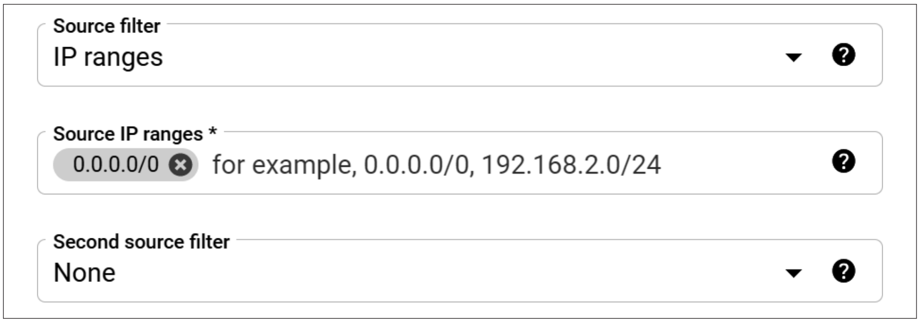


Targets
Specified target tags

Target tags *
allow-tcp-8080

Figure 12-1. Firewall rule: target tags

10. For an ingress rule, specify the source filter and use 0.0.0.0/0 for a source that allows traffic from any network, as shown in Figure 12-2.



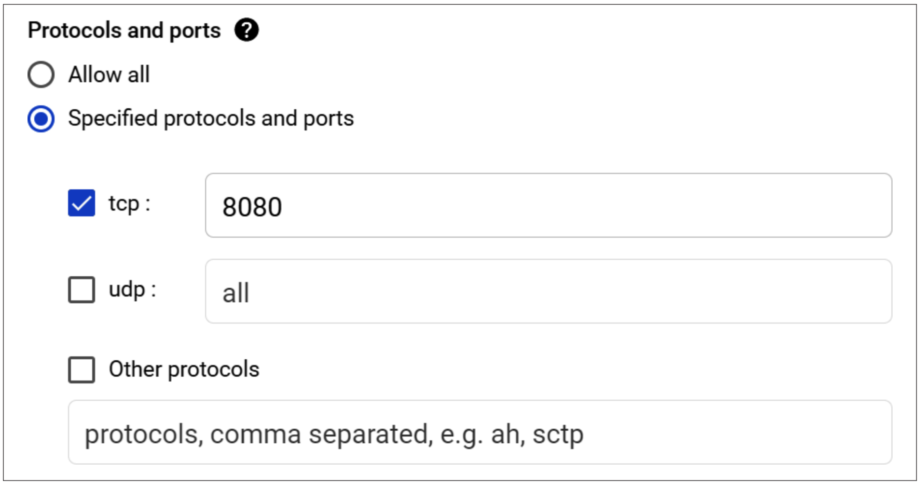
Source filter
IP ranges

Source IP ranges *
0.0.0.0/0 for example, 0.0.0.0/0, 192.168.2.0/24

Second source filter
None

Figure 12-2. Firewall rule: allow all traffic

11. In Protocols And Ports, define those to which the rule applies, as shown in Figure 12-3.


Protocols and ports

Allow all [✓ Specified protocols and ports]

[✓ tcp :] 8080

udp : all

Other protocols

protocols, comma separated, e.g. ah, sctp

Figure 12-3. Firewall rule: specified ports

12. Click Create.

Discussion

Firewall rules protect your virtual machine instances and are abstracted from the configuration of the instance. This means your instance can be a Windows or Linux, and the firewall rules will accept or deny connections to or from your virtual machine no matter the configuration.

12.4 Serving Content for Users in a Specific Region

Problem

You want to serve web content to users that is relevant to where they are. For example, a user is browsing from New Jersey, and you want to serve products that are currently in stock in your physical New Jersey store locations. You also want to do the same for all stores globally.

Solution

Use the custom headers available on the Google Cloud external HTTP(s) load balancer to add the required client connection information, which includes the client_city. This allows your web application to serve the content for the users in their respective region. In this recipe, you create a GKE cluster, deploy a sample application, expose the application, and, finally, add custom headers to the request to serve content for the user's region.

1. Set your project ID as environment variable for the `gcloud` command:

   ```
   gcloud config set project $PROJECT_ID
   ```

2. Set the Google Cloud zone for the cluster:

   ```
   gcloud config set compute/zone us-west1-a
   ```

3. Create a cluster named **`hello-cluster`**:

   ```
   gcloud container clusters create hello-cluster
   ```

4. Ensure that you are connected to your GKE cluster and replace COMPUTE_ZONE with the zone you entered in step 2:

   ```
   gcloud container clusters get-credentials hello-cluster --zone COM-
   PUTE_ZONE
   ```

5. For the manifest files, you can either create the file with the provided code or access *deployment.yaml* in the GitHub repository for this book.

6. Create a *deployment.yaml* manifest:

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: web
     namespace: default
   spec:
     selector:
       matchLabels:
         run: web
     template:
       metadata:
         labels:
           run: web
       spec:
         containers:
         - image: us.gcr.io/ruicosta-blog/header-app:1.3.0
           imagePullPolicy: IfNotPresent
           name: web
           ports:
           - containerPort: 8080
             protocol: TCP
   ```

7. Create a *service.yaml* manifest:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: web
     namespace: default
   spec:
     ports:
     - port: 8080
       protocol: TCP
       targetPort: 8080
   ```

```
    selector:
      run: web
    type: NodePort
```

8. Create an *ingress.yaml* manifest:

   ```
   apiVersion: extensions/v1beta1
   kind: Ingress
   metadata:
     name: basic-ingress
   spec:
     backend:
       serviceName: web
       servicePort: 8080
   ```

9. Apply the resources to the cluster:

   ```
   kubectl apply -f deployment.yaml
   kubectl apply -f service.yaml
   kubectl apply -f ingress.yaml
   ```

10. Find the external IP address of the load balancer:

    ```
    kubectl get ingress basic-ingress
    ```

11. Note the external IP address and open a browser window to view your deployed application. It will take a few minutes for the load balancer to be ready; you will get errors such as 404 or 500 until the load balancer is available globally.

12. To request the external HTTP(s) load balancer to add custom headers to the request and responses, run the following command:

    ```
    gcloud compute backend-services update \
    BACKEND_SERVICE_NAME \
    --global \
    --custom-request-header='X-PLACE:{client_city}'
    ```

    This command will create a custom header with the originating location of the user, which you can access to perform some business logic for your application.

13. Reload the browser window to view the value of the originating city.

Discussion

In this recipe, you added a custom header to your requests and responses so you can access the values from your web application to perform some business logic. Before creating the custom headers, you first deployed a sample application to Kubernetes with the declarative YAML files you created in steps 7 through 9. To learn more about creating GKE clusters, please review Chapter 6. The headers can include additional information such as client latency, transport layer security (TLS) parameters, and client connection information. Try adding new custom headers to collect the following variables as a challenge task (referenced from the backend services overview).

client_rtt_msec
Estimated round-trip transmission time between the load balancer and the HTTP(S) client, in milliseconds.

client_region
The country (or region) associated with the client's IP address.

client_city
Name of the city from which the request originated, for example, Mountain View for Mountain View, California.

12.5 Configuring VPC Network Peering

Problem

You want to deploy an ingress gateway to your GKE cluster, using native Kubernetes resources, and not manage an ingress controller.

Solution

You can use the Gateway API and the GKE Gateway controller (GCP's implementation of the Gateway API) as an ingress gateway to your GKE cluster.

1. Enable APIs:

   ```
   gcloud services enable --project ${PROJECT_ID} \
                  cloudresourcemanager.googleapis.com \
                  compute.googleapis.com \
                  container.googleapis.com
   ```

2. Create a GKE cluster:

   ```
   gcloud beta container --project $PROJECT_ID \
       clusters create "sample-cluster" \
       --zone "us-central1-c" --enable-ip-alias \
       --cluster-version "1.20.6-gke.1000" \
       --release-channel "rapid"
   ```

3. Create a *v1-service.yaml* manifest with the following code:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: hello-world-v1-service
   spec:
     ports:
     - port: 8080
       targetPort: 8080
     selector:
       app: hello-world-v1
   ```

4. Deploy the *v1-service.yaml* manifest:

```
kubectl apply -f v1-service.yaml
```

5. Create a *v2-service.yaml* manifest with the following code:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world-v2-service
spec:
  ports:
  - port: 8080
    targetPort: 8080
  selector:
    app: hello-world-v2
```

6. Deploy the *v2-service.yaml* manifest:

```
kubectl apply -f v2-service.yaml
```

7. Create a *v1-deployment.yaml* manifest with the following code:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world-v1
      version: v1
  template:
    metadata:
      labels:
        app: hello-world-v1
        version: v1
    spec:
      containers:
      - name: whereami
        image: gcr.io/google-samples/whereami:v1.1.3
        ports:
          - containerPort: 8080
        env:
        - name: METADATA
          value: "hello-world-v1"
```

8. Deploy the *v1-deployment.yaml* manifest:

```
kubectl apply -f v1-deployment.yaml
```

9. Create a *v2-deployment.yaml* manifest with the following code:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-v2
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world-v2
      version: v2
  template:
    metadata:
      labels:
        app: hello-world-v2
        version: v2
    spec:
      containers:
      - name: whereami
        image: gcr.io/google-samples/whereami:v1.1.3
        ports:
          - containerPort: 8080
        env:
        - name: METADATA
          value: "hello-world-v2"
```

10. Create a *gateway.yaml* manifest with the following code:

```
kind: Gateway
apiVersion: networking.x-k8s.io/v1alpha1
metadata:
  name: external-http
spec:
  gatewayClassName: gke-l7-gxlb
  listeners:
  - protocol: HTTP
    port: 80
    routes:
      kind: HTTPRoute
      selector:
        matchLabels:
          gateway: external-http
```

11. Deploy the *gateway.yaml* manifest:

```
kubectl apply -f v1-deployment.yaml
```

12. Create a *v1-http-route.yaml* manifest with the following code:

```
kind: HTTPRoute
apiVersion: networking.x-k8s.io/v1alpha1
metadata:
  name: hello-world-route-v1
  labels:
    gateway: external-http
spec:
  rules:
  - matches:
    - path:
        value: /hellov1
    forwardTo:
    - serviceName: hello-world-v1-service
      port: 8080
```

13. Deploy the *v1-http-route.yaml* manifest:

```
kubectl apply -f v1-http-route.yaml
```

14. Create a *v2-http-route.yaml* manifest with the following code:

```
kind: HTTPRoute
apiVersion: networking.x-k8s.io/v1alpha1
metadata:
  name: hello-world-route-v2
  labels:
    gateway: external-http
spec:
  rules:
  - matches:
    - path:
        value: /hellov2
    forwardTo:
    - serviceName: hello-world-v2-service
      port: 8080
```

15. Deploy the *v2-http-route.yaml* manifest:

```
kubectl apply -f v2-http-route.yaml
```

16. Run `kubectl describe gateway` to retrieve the public IP of your gateway.

In your web browser, you can now access the application as *GATEWAYIP/hellov1* and *GATEWAYIP/hellov2*.

Discussion

The Gateway API is an API in the Kubernetes system and aims to standardize ingress into clusters. In this recipe, we walked through how to provide a multitenant gateway that routes users to two deployments, depending on the HTTP route. The Gateway API provides many more features for advanced routing and traffic management, such as canarying, header-based routing, and more.

12.6 Creating VPN Gateways with Cloud Routers

Problem

You want to establish a secure connection from your on-premises network to Google Cloud.

Solution

Create a VPN gateway from a simulated on-premises network to a Google Cloud VPC network to secure incoming and outgoing traffic between your two networks.

1. Create two VPC networks with the following parameters. (Refer to Recipe 12.1 as a guide to creating a VPC.)

   First VPC:

Property	Value
Name	cloud-vpc
Description	Enter an optional description

Property	Value
Name	cloud-vpc-subnet
Region	us-east1
IP address range	10.10.10.0/24

   After creating the first VPC labeled as cloud-vpc, create the second VPC, labeled *onprem-vpc*, which will simulate the on-premises network.

   Second VPC:

Property	Value
Name	onprem-vpc
Description	Enter an optional description

Property	Value
Name	onprem-vpc-subnet
Region	us-central1
IP address range	10.10.20.0/24

2. Create the defined firewalls in both VPC networks. You can use Recipe 12.3 as a guide to create firewall rules.

First Firewall Rule:

Property	Value
Name	allow-ssh-cloud-vpc
Network	cloud-vpc
Targets	All instances in the network
Source filter	IP ranges
Source IP ranges	0.0.0.0/0
Protocols and ports	Specified protocols and ports, and then *check* tcp, *type:* 22

Second Firewall Rule:

Property	Value
Name	allow-ssh-onprem-vpc
Network	onprem-vpc
Targets	All instances in the network
Source filter	IP ranges
Source IP ranges	0.0.0.0/0
Protocols and ports	Specified protocols and ports, and then *check* tcp, *type:* 22

3. In the Google Cloud Console, open Hybrid Connectivity and choose Cloud Routers.
4. Click Create Router.
5. Use the following parameters for the options on the Create Router page:

Property	Value
Name	cloud-vpc-cloud-router
Network	cloud-vpc
Region	us-east1
Google ASN	65470

6. Create the simulated on-premises router. In the Google Cloud Console, open Hybrid Connectivity and choose Cloud Routers.
7. Click Create Router.

8. Use the following parameters for the options on the Create Router page:

Property	Value
Name	onprem-vpc-cloud-router
Network	onprem-vpc
Region	us-central1
Google ASN	65503

9. Each gateway needs a static External IP address. In the Google Cloud Console, choose VPC Network and click External IP Addresses.
10. Click Reserve Static Address.
11. Use the following parameters for the options in the Reserve Static Address page:

Property	Value
Name	cloud-vpc-ip
Type	Regional
Region	us-east1
Attached to	None

12. Reserve a second IP address for the simulated on-premises network with the following values:

Property	Value
Name	onprem-vpc-ip
Type	Regional
Region	us-central1
Attached to	None

13. Create the VPN tunnel from the cloud VPC network to the simulated on-premises network. In the Google Cloud Console, click Hybrid Connectivity and choose VPN.
14. Click the Create VPN connection.
15. Use the following values for the VPN connection:

Property	Value
Name	vpn-1
Network	cloud-vpc
Region	us-east1
IP address	cloud-vpc-ip

For the tunnels:

Property	Value
Remote peer IP address	Enter the reserved on-prem IP address
IKE version	IKEv2
Shared secret	googlecloud
Routing options	Dynamic (Border Gateway Protocol [BGP])
Cloud router	cloud-vpc-cloud-router

BGP:

Property	Value
Name	bgp1to2
Peer ASN	65503
Cloud Router BGP IP	169.254.0.1
BGP peer IP	169.254.0.2

16. Create the VPN tunnel from the simulated on-premises network to the cloud VPC network. In the Google Cloud Console, click Hybrid Connectivity and choose VPN.
17. Click the Create VPN connection.
18. Use the following values for the VPN connection:

Property	Value
Name	vpn-2
Network	onprem-vpc
Region	us-central1
IP address	onprem-vpc-ip

For the tunnels:

Property	Value
Remote peer IP address	Enter the reserved cloud IP address
IKE version	IKEv2
Shared secret	googlecloud
Routing options	Dynamic (BGP)
Cloud router	onprem-vpc-cloud-router

BGP:

Property	Value
Name	bgp1to2
Peer ASN	65503
Cloud Router BGP IP	169.254.0.2
BGP peer IP	169.254.0.1

After a few minutes, you will see the VPN tunnels with green checkmarks, informing you that the connections have been established successfully.

Discussion

The Google Cloud VPN connects your on-premises network to your Google Cloud VPC network through an IPsec connection to encrypt the traffic. Keep in mind that the connection is not limited to on-premises networks; any peer network can work as long as it meets the requirements of the cloud VPN. All traffic passing through the VPN connection will be encrypted.

12.7 Deployments of Networks Using Terraform

Problem

You want to set up a method of automating the creation of VPC networks with associated resources as firewall rules.

Solution

You will use Terraform and the associated manifest files to automate the deployment of resources in Google Cloud.

1. Git clone this book's GitHub repository.
2. In your Google Cloud Shell or your local workstation, download and install Terraform.
3. In the cloned repository, go to the *12-networking/12-07* folder and run the following command to initialize Terraform:
   ```
   terraform init
   ```
4. Create an execution plan by running the following command:
   ```
   terraform plan
   ```
5. Apply the desired changes by running the following command:
   ```
   terraform apply
   ```

6. Enter *yes* to continue.
7. In the Google Cloud Console, from the navigation menu, choose VPC Network and then click VPC Networks.

You should see your newly created VPC network, which is defined in the Terraform manifest files.

Discussion

By using Terraform with Google Cloud, you can automate the deployment of cloud resources and validate that the respective rules are followed based on the declarations in the manifest files. To learn more about Terraform, please visit *https://www.terraform.io*.

12.8 Limiting Access to Only Authorized Networks with VPC Service Controls

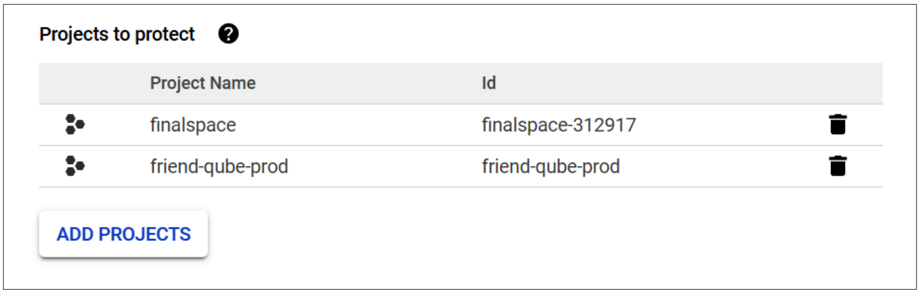
Problem

You need a method to prevent data exfiltration from your Google Cloud resources.

Solution

Using VPC Service Controls, you can limit access to only authorized networks, restricting users from copying data outside of the perimeter you defined.

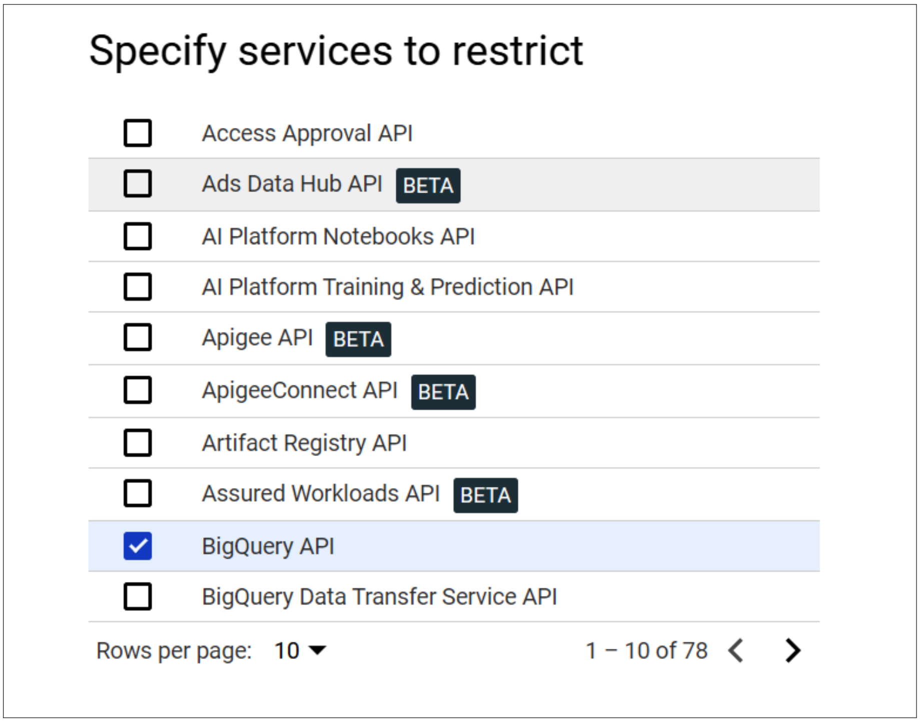
1. In your Google Cloud Console, open Security and choose VPC Service Controls.
2. Make sure you have chosen an organization rather than a project. VPC Service Controls are set at the organizational level.
3. Click New Perimeter.
4. On the New VPC Service Perimeter page, enter a name for the perimeter.
5. Select the projects that you want to secure within the perimeter, as shown in Figure 12-4:
    a. Click the Add Projects button.
    b. Select that project's checkbox.


Projects to protect

	Project Name	Id	
	finalspace	finalspace-312917	
	friend-qube-prod	friend-qube-prod	

Figure 12-4. Google Cloud projects

6. Select the services you want to secure, as shown in Figure 12-5.


Specify services to restrict

☐	Access Approval API
☐	Ads Data Hub API BETA
☐	AI Platform Notebooks API
☐	AI Platform Training & Prediction API
☐	Apigee API BETA
☐	ApigeeConnect API BETA
☐	Artifact Registry API
☐	Assured Workloads API BETA
✓	BigQuery API
☐	BigQuery Data Transfer Service API

Figure 12-5. Services to restrict

7. Click Save.

The service perimeter may take up to 30 minutes to propagate and take effect.

Discussion

VPC Service Controls protect your projects from data exfiltration by creating a perimeter that only allows access to authorized networks. For example, VPC Service Controls can prevent applications and users from reading data from or copying data to a resource outside the perimeter.

Index

About the Authors

Rui Costa has worked at Google in various roles, most recently as a learning consultant working with strategic customers and partners to create and execute on their custom Google Cloud learning plans. Rui has served as an AI coach for the Google AI Impact Challenge, where he had the opportunity to help organizations using AI to address societal challenges. Rui is also the founder of the Speech Analysis Framework, which has successfully graduated to become a product within Google and holds a defensive publication, "Secure Sharing of Pre-trained Machine Learning Models For Hands-on Training At Scale."

You can find Rui on Twitter and LinkedIn.

Drew Hodun has had a number of roles at Google, mostly customer-facing and focused on machine learning applications running on Google Cloud. He's worked in the autonomous vehicle, financial services, and media and entertainment industries. Most recently, Drew has been a tech lead on a public-sector machine learning product team. He has spoken at a variety of conferences, including Predictive Analytics World, Google Cloud Next, O'Reilly AI Conference, and others.

You can find Drew on Twitter and LinkedIn.

Colophon

The animal on the cover of *Google Cloud Cookbook* is the asian golden eagle (*aquila chrysaetos daphanea*), also known as the berkut. This subspecies can be found in Kazakhstan, Iran, Manchuria and China, and along the Himalayas from Pakistan to Bhutan. This bird of prey is recognized by its dark feathers, a dark forehead and crown, and a brown-red neck plume.

Golden eagles use their powerful feet and massive, sharp talons, to snatch up a variety of prey (mainly hares, rabbits, marmots, and squirrels). Notably, they can carry up to 8 lbs in flight and fly up to 80 mph, with maximum speeds of 200 mph during a dive.

Direct and indirect human-caused mortality, disturbance, and elimination of prey limit golden eagle populations. Recreational activities may disturb breeding, as golden eagles are likely to abandon nests during incubation if they are disturbed. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *Gardens and Menagerie of the Zoological Society*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.