Chapter 9. MLOps for GCP

By Noah Gift

> *The best bonsai teachers have both a mastery of the reality and an ability not only to explain but to inspire. John once said, "Jin this part and wire it so it will dry with a nice shape." "What shape?" I asked. "You decide," he replied, "it's not for me to sing your song for you!"*
>
> —Dr. Joseph Bogen

The Google Cloud Platform (GCP) is unique compared to its competitors. On the one hand, it has been marginally enterprise-focused; on the other, it has world-class research and development that has created category-leading technology, including products like Kubernetes and TensorFlow. Yet one more unique aspect of the Google Cloud is the rich collection of educational resources available to students and working professionals through *https://edu.google.com*.

Let's dive into the Google Cloud with an emphasis on using it to perform MLOps.

Google Cloud Platform Overview

Every cloud platform has pros and cons, so let's start by covering the three main cons of the Google Cloud Platform. First, with Google trailing AWS and Microsoft Azure, one disadvantage of using Google is that it has fewer certified practitioners. In Figure 9-1, you can see that in 2020, AWS and Azure controlled over 50% of the market, and Google Cloud was less than 9%. Hiring talent for the Google Cloud Platform is more challenging as a result.

A second disadvantage is that Google is part of what Harvard Professor Shoshana Zuboff calls surveillance capitalism, in which "Silicon Valley and other corporations are mining users' information to predict and

shape their behavior." Thus, it is theoretically possible that technology regulation could impact market share in the future.

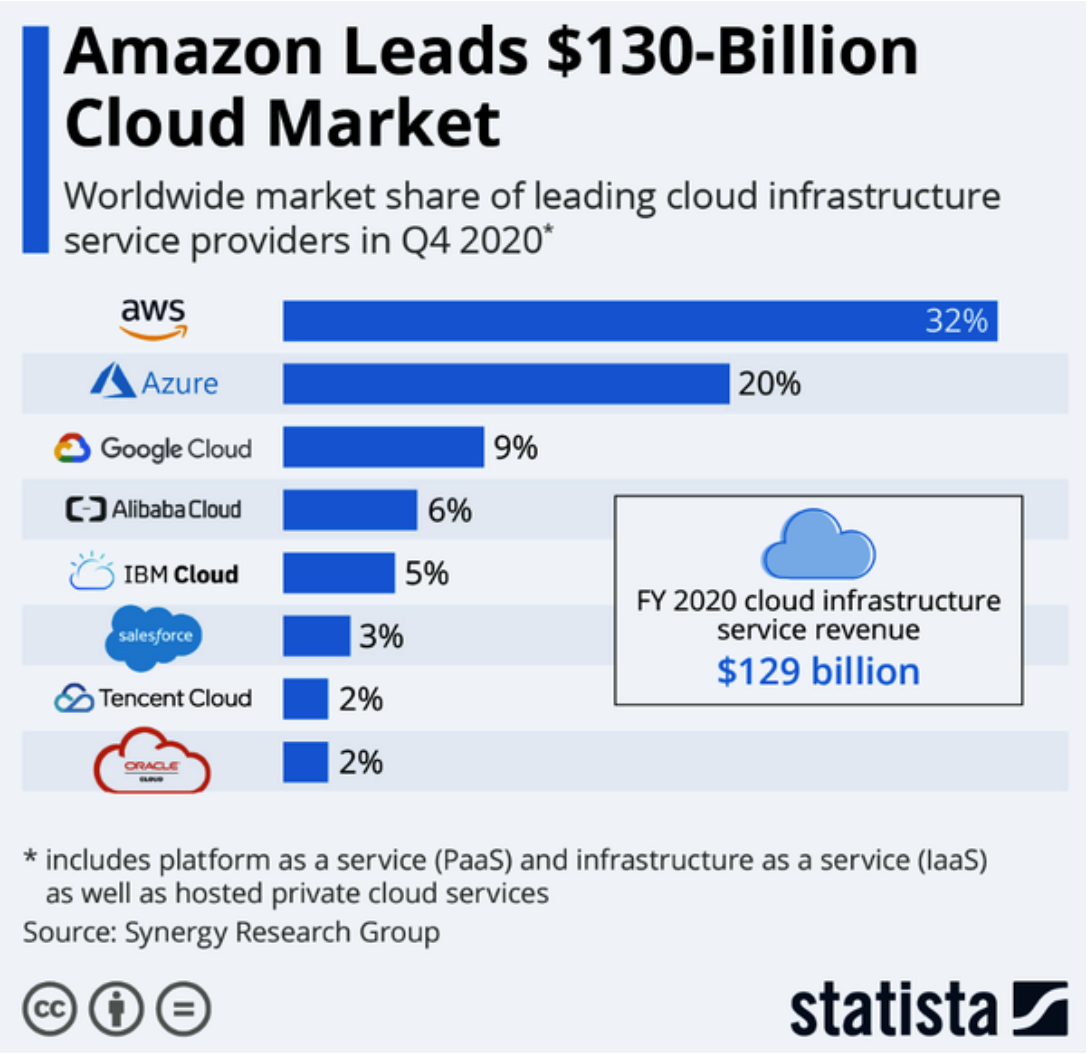

Figure 9-1. GCP Cloud market share

Finally, Google has a reputation for a poor user and customer experience and frequently abandons products like Google Hangouts and the Google Plus social network. Could it then discontinue Google Cloud if it remains the third-best option in the next five years?

While these are substantial challenges, and Google would be wise to address the cultural issues that led to these cons quickly, there are many unique advantages to the Google platform due to its culture. For example, while AWS and Microsoft are customer service-oriented cultures with a rich history of enterprise customer support, Google famously didn't have phone support for most products. Instead, its culture focuses on intense "leet code" style interviews to only "hire the best." Additionally, research and development of mind-numbingly complex solutions that work at "planet-scale" is something it does well. In particular, three of Google's

most successful open source projects show this cultural strength: Kubernetes, the Go language, and the deep learning framework TensorFlow.

Ultimately the number one advantage of using the Google Cloud may be that its technology is ideal for a multicloud strategy. Technologies like Kubernetes and Tensor Flow work well on any cloud and are widely adopted. As a result, using Google Cloud could be a hedge for large companies wanting to check the power of their vendor relationship with either AWS or Azure. Additionally, these technologies have wide adoption, so it is relatively straightforward to hire for positions that require expertise in TensorFlow.

Let's take a look at Google Cloud's core offerings. These services divide into four clean categories: Compute, Storage, Big Data, and Machine Learning, as shown in Figure 9-2.

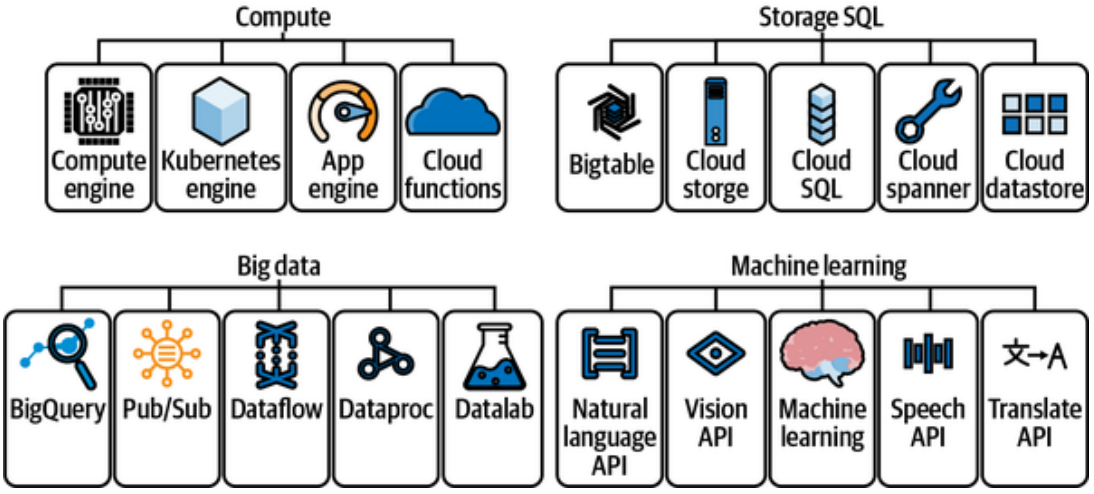

Figure 9-2. GCP Cloud services

Let's define the major components of Google Cloud next, starting with Compute:

Compute Engine

> Like other cloud vendors (notably AWS and Azure), GCP offers virtual machines as a service. Compute Engine is a service that enables you to create and run virtual machines on Google's infrastructure. Perhaps the most critical takeaway is that there are many different virtual machines, including Compute Intensive, Memory Intensive, Accelerator Optimized, and General Purpose.

Additionally, there are preemptible VMs, available for up to 24 hours, suited for batch jobs, and they can save up to 80% on storage costs.

As an MLOps practitioner, it is critical to use suitable types of machines for the task at hand. Costs do matter in the actual world, and the ability to accurately forecast costs could make or break a company doing machine learning. For example, with deep learning, it could be optimal to use Accelerator Optimized instances since they could leverage the additional massively parallel capabilities of NVIDIA GPUs. On the other hand, it would be incredibly wasteful to use these instances for machine learning training that cannot take advantage of GPUs. Similarly, by architecting around preemptible VMs for batch machine learning, an organization could save up to 80% of the cost.

Kubernetes Engine and Cloud Run

Since Google created Kubernetes and maintains it, the support for doing work on Kubernetes is excellent through its GKE (Google Kubernetes Engine). Alternatively, Cloud Run is a high-level service that abstracts away many of the complexities of running containers. Cloud Run is a good starting point for the Google Cloud Platform for organizations wanting a simple way to deploy containerized machine learning applications.

App Engine

Google App Engine is a fully managed PaaS. You can write code in many languages, including Node.js, Java, Ruby, C#, Go, Python, or PHP. MLOps workflows could use App Engine as the API endpoint of a fully automated continuous delivery pipeline using GCP Cloud Build to deploy changes.

Cloud Functions

Google Cloud Functions act as a FaaS (functions as a service). FaaS works well with an event-driven architecture. For example, Cloud Functions could trigger a batch machine learning training job or deliver an ML prediction in response to an event.

Next, let's talk about storage on Google Cloud. Concerning MLOps, the main option to discuss is its Cloud Storage product. It offers unlimited storage, worldwide accessibility, low latency, geo-redundancy, and high durability. These facts mean that for MLOps workflows, the data lake is the location where unstructured and structured data resides for batch processing of machine learning training jobs.

Closely associated with this offering are the big data tools available from GCP. Many offerings assist in moving, querying, and computing big data. One of the most popular offerings is Google BigQuery because it offers a SQL interface, serverless paradigm, and the ability to do machine learning within the platform. Google BigQuery is a great place to start doing machine learning on GCP because you can solve the entire MLOps value chain from this one tool.

Finally, the machine learning and AI capabilities coordinate in a product called Vertex AI. One advantage of Google's approach is that it aims to be an MLOps solution from the beginning. The workflow of Vertex AI allows a structured approach to ML, including the following:

- Dataset creation and storage
- Training an ML model
- Storing the model in Vertex AI
- Deploying the model to an endpoint for prediction
- Testing and creating prediction requests
- Using traffic splitting for endpoints
- Managing the lifecycle of ML models and endpoints

According to Google, these capabilities factor into how Vertex AI enables MLOps, as shown in Figure 9-3. At the center of these seven components is data and model management, the central element of MLOps.

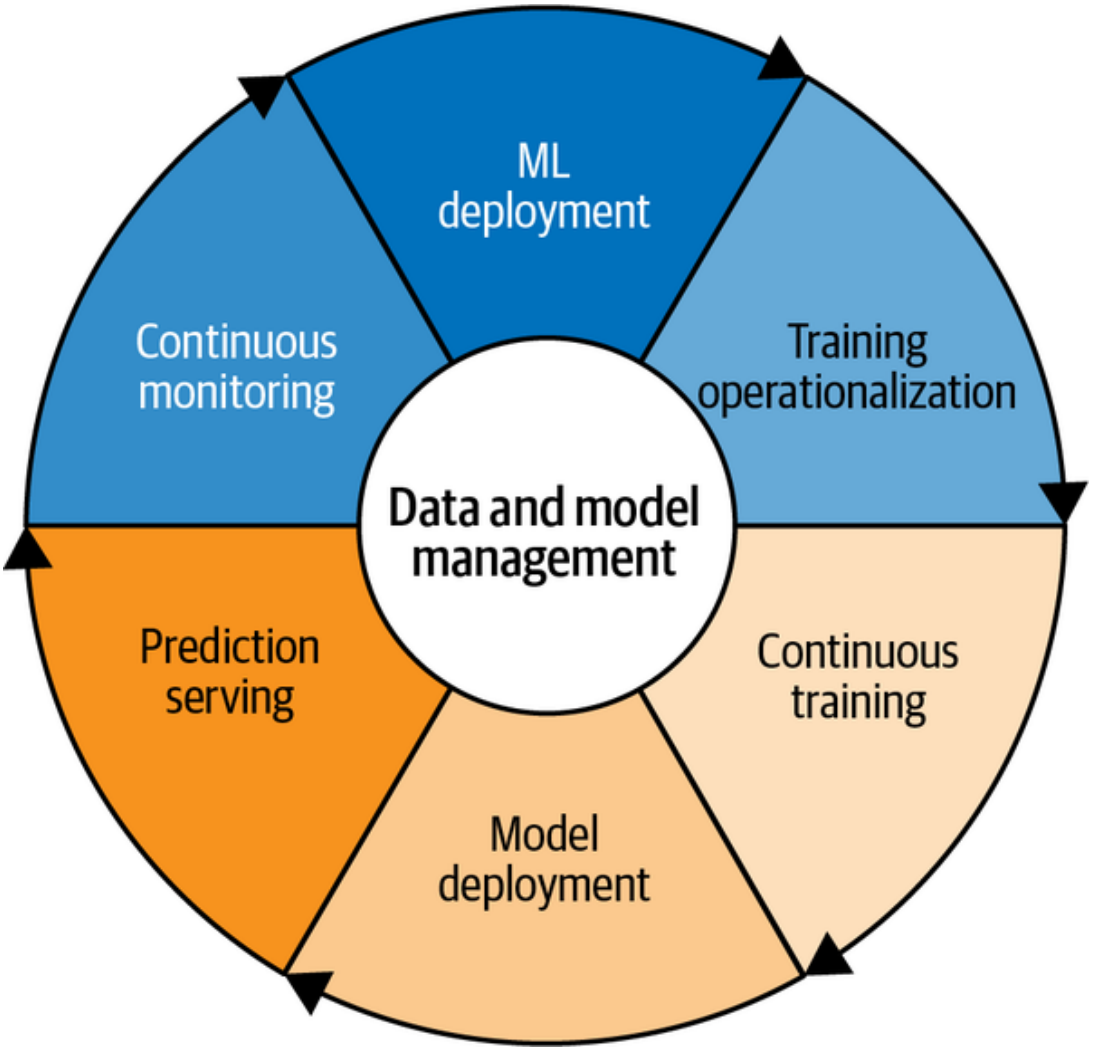

Figure 9-3. Google's seven components of MLOps

This thought process culminates in Google's idea of end-to-end MLOps, as described in Figure 9-4. A comprehensive platform like Vertex AI enables a comprehensive way to manage MLOps.

In a nutshell, MLOps on the Google Cloud Platform is straightforward due to Vertex AI and subcomponents of this system like Google BigQuery. Next, let's get into more detail on CI/CD on GCP, a nonoptional foundational component of MLOps.

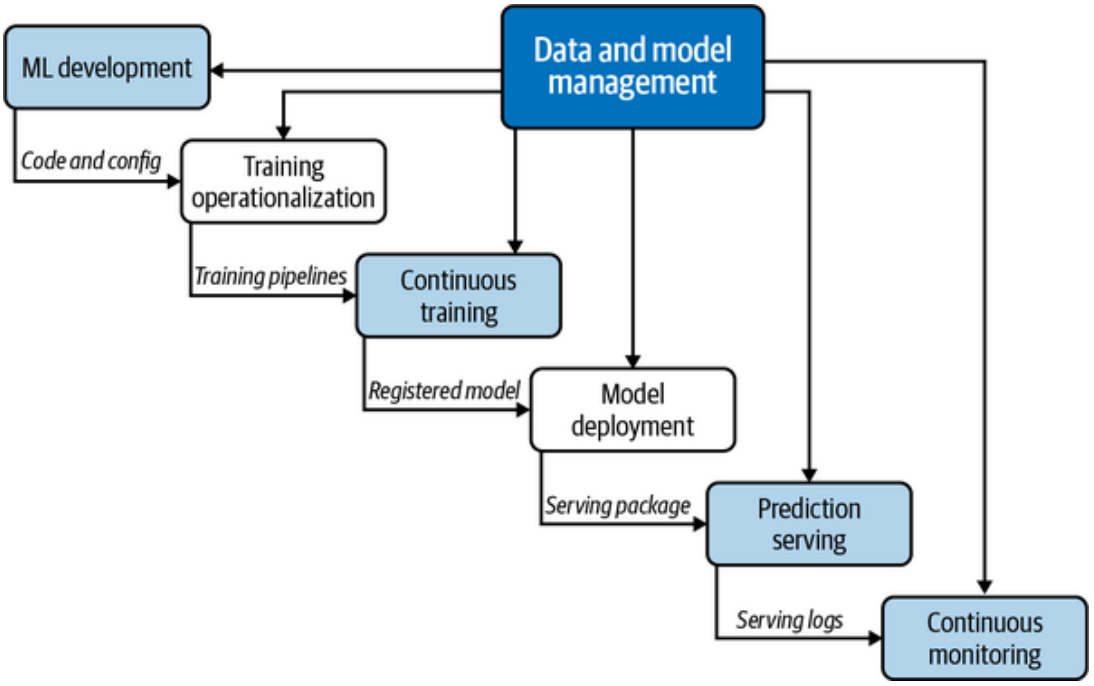

Figure 9-4. End-to-end MLOps on GCP

Continuous Integration and Continuous Delivery

One of the most important and yet neglected areas of a project involves continuous integration. Testing is a fundamental component to doing both DevOps and MLOps. For GCP, there are two main continuous integration options: use a SaaS offering like GitHub Actions or use the cloud native solution Cloud Build. Let's take a look at both options. You can view an entire starter project scaffold in this gcp-from-zero GitHub repository.

First, let's take a look at Google Cloud Build. Here is an example of the configuration file for Google Cloud Build, *cloudbuild.yaml*:

```
steps:
- name: python:3.7
  id: INSTALL
  entrypoint: python3
  args:
  - '-m'
  - 'pip'
  - 'install'
  - '-t'
  - '.'
  - '-r'
  - 'requirements.txt'
- name: python:3.7
```

```
    entrypoint: ./pylint_runner
    id: LINT
    waitFor:
    - INSTALL
  - name: "gcr.io/cloud-builders/gcloud"
    args: ["app", "deploy"]
  timeout: "1600s"
  images: ['gcr.io/$PROJECT_ID/pylint']
```

A recommended way to work with Google Cloud is to use the built-in editor alongside the terminal, as shown in Figure 9-5. Note that a Python virtual environment is activated.

Figure 9-5. GCP editor

One takeaway is that Google Cloud Build is a bit clunky with testing and linting code compared to GitHub Actions, but it does make deploying services like Google App Engine easy.

Now let's look at how GitHub Actions works. You can reference the *python-publish.yml* configuration file:

```
name: Python application test with GitHub Actions

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
```

```
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.8
      uses: actions/setup-python@v1
      with:
        python-version: 3.8
    - name: Install dependencies
      run: |
        make install
    - name: Lint with pylint
      run: |
        make lint
    - name: Test with pytest
      run: |
        make test
    - name: Format code
      run: |
        make format
```

A critical difference between the two approaches is that GitHub focuses on delivering an incredible developer experience, while GCP focuses on the cloud experience. One strategy is to use GitHub Actions for developer feedback, i.e., linting and testing code, and to use Google Cloud Build for deployment.

With a handle on CI/CD systems on GCP, let's explore a core Google compute technology, Kubernetes.

Kubernetes Hello World

One way of thinking about Kubernetes is as a "mini-cloud" or a "cloud-in-a-box." Kubernetes allows for the creation of near-infinite application complexity. A few of the capabilities of Kubernetes that make it ideal for MLOps include:

- High-availability architecture
- Autoscaling
- Rich ecosystem
- Service discovery
- Container health management

- Secrets and configuration management
- Kubeflow (end-to-end ML platform for Kubernetes)

Figure 9-6 notes that ML frameworks from TensorFlow to scikit-learn coordinate on top of the core Kubernetes architecture. Finally, Kubernetes, as discussed earlier, can run in many clouds or in your own data center.

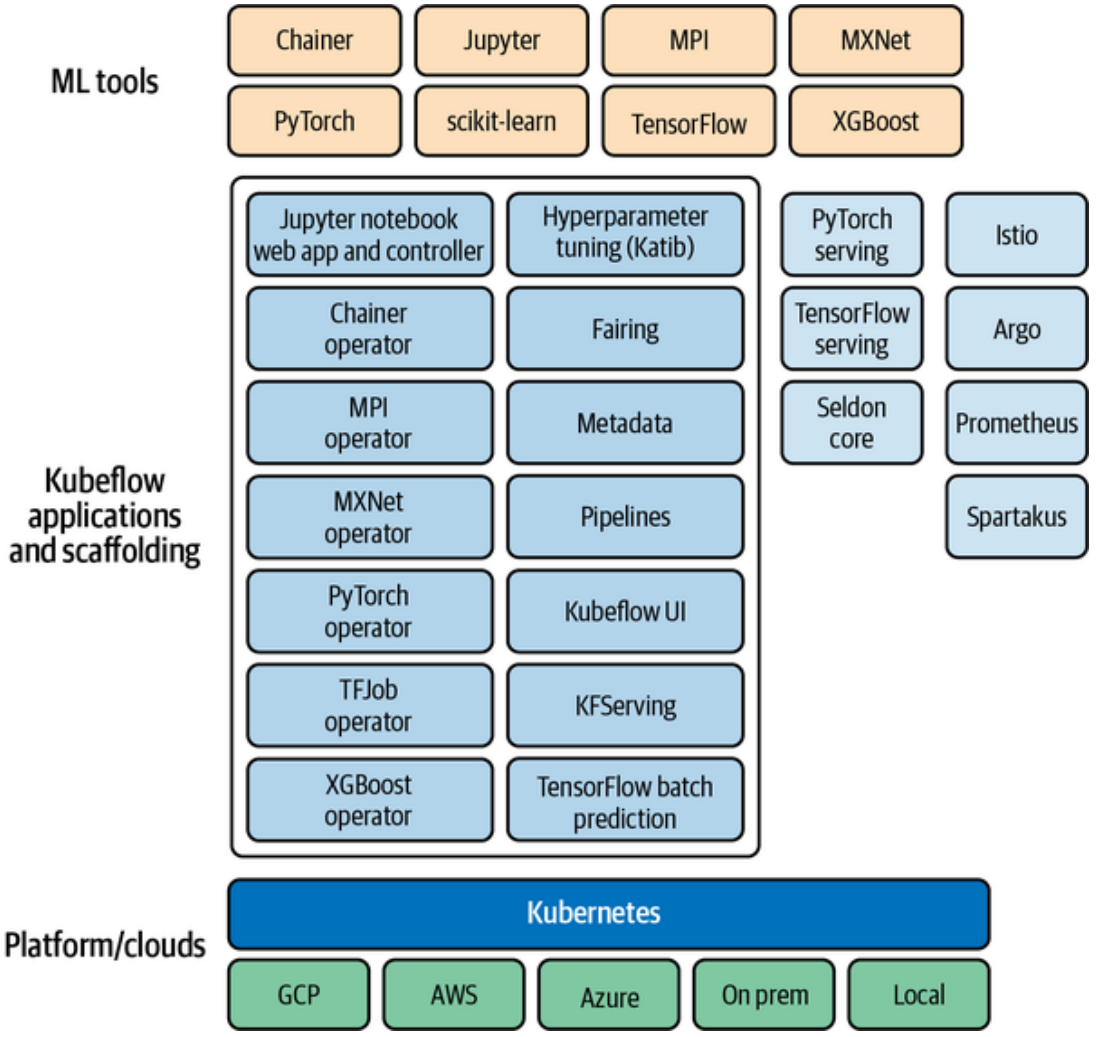

Figure 9-6. Kubeflow architecture

The foundation of the Kubernetes architecture in Figure 9-7 shows the core operations involved in Kubernetes include the following:

- Creating a Kubernetes cluster.
- Deploying an application into the cluster.
- Exposing application ports.
- Scaling an application.
- Updating an application.

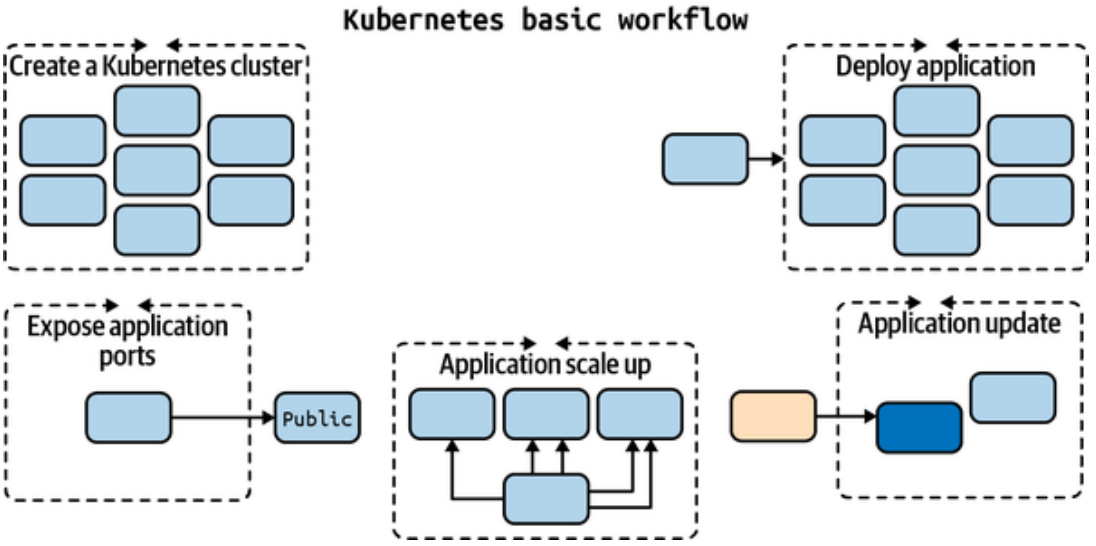

Figure 9-7. Kubernetes basics

The Kubernetes hierarchy in Figure 9-8 shows a Kubernetes control node that manages the other nodes containing one or more containers inside a pod.

Figure 9-8. Kubernetes hierarchy

NOTE

There are two main methods: set up a local cluster (preferably with Docker Desktop) or provision a cloud cluster: Amazon through Amazon EKS, Google through Google Kubernetes Engine GKE, and Microsoft through Azure Kubernetes Service (AKS).

One of the "killer" features of Kubernetes is the ability to set up autoscaling via the Horizontal Pod Autoscaler (HPA). The Kubernetes HPA will automatically scale the number of pods (remember they can contain multiple containers) in a replication controller, deployment, or replica set. The scaling uses CPU utilization, memory, or custom metrics defined in the Kubernetes Metrics Server.

In Figure 9-9, Kubernetes is using a control loop to monitor metrics for the cluster and perform actions based on the metrics received.

Figure 9-9. Kubernetes autoscaler

Since Kubernetes is a core strength of the Google Platform and how much of MLOps works on the platform, let's dive right into a "hello world" Kubernetes example. This project uses a simple Flask app that returns correct change as the base project and converts it to Kubernetes. You can find the complete source in the repository on GitHub.

In Figure 9-10 the Kubernetes nodes attach to the Load Balancer.

Figure 9-10. Kubernetes hello world

Let's look at the assets in the repository.

- *Makefile*: Builds project
- *Dockerfile*: Container configuration
- *app.py*: Flask app
- *kube-hello-change.yaml*: Kubernetes YAML Config

To get started, do the following steps:

1. Create Python virtual environment:

```
python3 -m venv ~/.kube-hello && source ~/.kube-hello/bin/activate
```

2. Run `make all` to execute multiple build steps, including installing the libraries, linting the project, and running the tests.

MAKING USE OF CERTIFIED CONTAINERS

One of the Docker workflow advantages for developers is using certified containers from the "official" development teams. In this diagram, a developer uses the official Python base image developed by the core Python developers. This step works using the `FROM` statement, which loads in a previously created container image.

As the developer changes to the Dockerfile, they test locally and then push the changes to a private Docker Hub repo. For example, in Figure 9-11, the changes can become available by a deployment process to a cloud or another developer.

Figure 9-11. Docker developer workflow

Next, build and run a Docker Container:

1. Install Docker Desktop
2. To build the image locally, do the following:

   ```
   docker build -t flask-change:latest .
   ```

   or run `make build`, which has the same command.
3. To verify container run `docker image ls`
4. To run the container do the following:

   ```
   docker run -p 8080:8080 flask-change
   ```

   or run `make run`, which has the same command.
5. In a separate terminal, invoke the web service via `curl`, or run `make invoke` which has the same command:

   ```
   curl http://127.0.0.1:8080/change/1/34
   ```

   Here's the output:

```
$ kubectl get nodes
[
  {
    "5": "quarters"
  },
  {
    "1": "nickels"
  },
  {
    "4": "pennies"
  }
]
```

6. Stop the running Docker container by using Ctrl+C command

Next, run Kubernetes locally:

1. Verify Kubernetes is working via docker-desktop context:

```
(.kube-hello) ➜  kubernetes-hello-world-python-flask git:(main) kubectl \
get nodes

NAME             STATUS   ROLES    AGE   VERSION
docker-desktop   Ready    master   30d   v1.19.3
```

2. Run the application in Kubernetes using the following command, which tells Kubernetes to set up the load-balanced service and run it:

```
kubectl apply -f kube-hello-change.yaml
```

or run `make run-kube`, which has the same command.
You can see from the config file that a load balancer along with three nodes are the configured application:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-flask-change-service
spec:
  selector:
```

```
    app: hello-python
  ports:
  - protocol: "TCP"
    port: 8080
    targetPort: 8080
  type: LoadBalancer

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-python
spec:
  selector:
    matchLabels:
      app: hello-python
  replicas: 3
  template:
    metadata:
      labels:
        app: hello-python
    spec:
      containers:
      - name: flask-change
        image: flask-change:latest
        imagePullPolicy: Never
        ports:
        - containerPort: 8080
```

3. Verify the container is running:

```
kubectl get pods
```

Here is the output:

```
NAME                            READY   STATUS    RESTARTS   AGE
flask-change-7b7d7f467b-26htf   1/1     Running   0          8s
flask-change-7b7d7f467b-fh6df   1/1     Running   0          7s
flask-change-7b7d7f467b-fpsxr   1/1     Running   0          6s
```

4. Describe the load balanced service:

```
kubectl describe services hello-python-service
```

You should see output similar to this:

```
Name:                     hello-python-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=hello-python
Type:                     LoadBalancer
IP Families:              <none>
IP:                       10.101.140.123
IPs:                      <none>
LoadBalancer Ingress:     localhost
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30301/TCP
Endpoints:                10.1.0.27:8080,10.1.0.28:8080,10.1.0.29:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

5. Invoke the endpoint to `curl` it:

```
make invoke
```

In the next section, run the command `make invoke` to query the microservice. The output of that action is shown here:

```
curl http://127.0.0.1:8080/change/1/34
[
  {
    "5": "quarters"
  },
  {
    "1": "nickels"
  },
  {
    "4": "pennies"
  }
]
```

To clean up the deployment, run `kubectl delete deployment hello-python` .

The next step beyond this basic tutorial is to use either GKE (Google Kubernetes Engine), Google Cloud Run (container as a service), or Vertex AI to deploy a machine learning endpoint. You can use the Python MLOps Cookbook repo as a base to do this. The Kubernetes technology is an excellent foundation for building ML-powered APIs, and with GCP, there are many options available if you use Docker format containers from the start.

With the basics of computing on GCP out of the way, let's discuss how cloud native databases like Google BigQuery go a long way toward adopting MLOps.

Cloud Native Database Choice and Design

One of the crown jewels of the Google Cloud Platform is Google BigQuery, for a few reasons. One of the reasons is how easy it is to get started, and another reason is the widespread publicly available databases. A good list of Google BigQuery open datasets is available on this Reddit page. Finally, from an MLOps perspective, one of the killer features of Google BigQuery is the ability to train and host ML models inside the Google BigQuery platform.

In looking at Figure 9-12, notice that Google BigQuery is the center of an MLOps pipeline that can export products to both business intelligence and machine learning engineering, including Vertex AI. This MLOps workflow is made possible because of the DataOps (Operationalization of Data) inputs such as public datasets, streaming API, and the Google Dataflow product. The fact that Google BigQuery performs machine learning inline streamlines the processing of big datasets.

Figure 9-12. Google BigQuery MLOps workflow

An example of this workflow is shown in Figure 9-13, where after the ML modeling occurred in Google BigQuery, the results export to Google Data

Studio. The artifact created from BigQuery is K-means clustering analysis shown in this shareable report.

Figure 9-13. Google Data Studio K-means clustering

As a starting point for doing MLOps on the GCP platform, BigQuery is an optimal choice due to the platform's flexibility. Next, let's talk about DataOps and Applied Data Engineering on the GCP platform.

DataOps on GCP: Applied Data Engineering

Data is input necessary for building machine learning at scale, and as such, it is a critical aspect of MLOps. In one sense, the GCP has an almost unlimited amount of ways to automate data flow. This fact is due to the variety of computing and storage options available, including high-level tools like Dataflow.

For the sake of simplicity, let's use a serverless approach to data engineering using Cloud Functions. To do this, let's look at how Google Cloud Functions work and how they can serve double duty as both the ML solution via an AI API call or an MLOps pipeline using Cloud Functions talking to Google Pub/Sub.

Let's start with an intentionally simple Google Cloud Function that returns the correct change. You can find the complete example here.

To get started, open the Google Cloud Console, create a new Cloud Function, and paste the following code inside, as shown in Figure 9-14. You can also "untoggle" the "Require authentication" to "Allow unauthenticated invocations."

```
import json

def hello_world(request):

    request_json = request.get_json()
```

```
    print(f"This is my payload: {request_json}")
    if request_json and "amount" in request_json:
        raw_amount = request_json["amount"]
        print(f"This is my amount: {raw_amount}")
        amount = float(raw_amount)
        print(f"This is my float amount: {amount}")
    res = []
    coins = [1, 5, 10, 25]
    coin_lookup = {25: "quarters", 10: "dimes", 5: "nickels", 1: "pennies"}
    coin = coins.pop()
    num, rem = divmod(int(amount * 100), coin)
    res.append({num: coin_lookup[coin]})
    while rem > 0:
        coin = coins.pop()
        num, rem = divmod(rem, coin)
        if num:
            if coin in coin_lookup:
                res.append({num: coin_lookup[coin]})
    result = f"This is the res: {res}"
    return result
```

Figure 9-14. Google Cloud Function

To invoke via the `gcloud` command line do the following:

```
gcloud functions call changemachine --data '{"amount":"1.34"}'
```

To invoke via the `curl` command, you can use the following.

```
curl -d '{
    "amount":"1.34"
}'     -H "Content-Type: application/json" -X POST <trigger>/function-3
```

Another approach is to build a command line tool to invoke your endpoint:

```
#!/usr/bin/env python
import click
import requests
```

```
@click.group()
@click.version_option("1.0")
def cli():
    """Invoker"""


@cli.command("http")
@click.option("--amount", default=1.34, help="Change to Make")
@click.option(
    "--host",
    default="https://us-central1-cloudai-194723.cloudfunctions.net/change722",
    help="Host to invoke",
)
def mkrequest(amount, host):
    """Asks a web service to make change"""

    click.echo(
        click.style(
            f"Querying host {host} with amount: {amount}", bg="green", fg="white"
        )
    )
    payload = {"amount": amount}
    result = requests.post(url=host, json=payload)
    click.echo(click.style(f"result: {result.text}", bg="red", fg="white"))


if __name__ == "__main__":
    cli()
```

Finally, one more approach is to either upload your ML model to Vertex AI or call an existing API endpoint that performs computer vision, NLP, or another ML-related task. You can find the complete example on GitHub. In the following example, let's use a preexisting NLP API. You will also need to add two third-party libraries by editing the *requirements.txt* file included in the Google Cloud scaffolding (see Figure 9-15).

Figure 9-15. Add requirements

Paste this code into the *main.py* function in the Google Cloud Shell Console:

```python
import wikipedia

from google.cloud import translate


def sample_translate_text(
    text="YOUR_TEXT_TO_TRANSLATE", project_id="YOUR_PROJECT_ID", language="fr"
):
    """Translating Text."""

    client = translate.TranslationServiceClient()

    parent = client.location_path(project_id, "global")

    # Detail on supported types can be found here:
    # https://cloud.google.com/translate/docs/supported-formats
    response = client.translate_text(
        parent=parent,
        contents=[text],
        mime_type="text/plain",  # mime types: text/plain, text/html
        source_language_code="en-US",
        target_language_code=language,
    )
    print(f"You passed in this language {language}")
    # Display the translation for each input text provided
    for translation in response.translations:
        print("Translated text: {}".format(translation.translated_text))
    return "Translated text: {}".format(translation.translated_text)


def translate_test(request):
    """Takes JSON Payload {"entity": "google"}"""
    request_json = request.get_json()
    print(f"This is my payload: {request_json}")
    if request_json and "entity" in request_json:
        entity = request_json["entity"]
        language = request_json["language"]
        sentences = request_json["sentences"]
        print(entity)
        res = wikipedia.summary(entity, sentences=sentences)
        trans = sample_translate_text(
            text=res, project_id="cloudai-194723", language=language
        )
        return trans
```

```
        else:
            return f"No Payload"
```

To invoke the function, you can call it from the Google Cloud Shell:

```
gcloud functions call translate-wikipedia --data\
 '{"entity":"facebook", "sentences": "20", "language":"ru"}'
```

You can see the output of the Russian translation in the Google Cloud Shell terminal in Figure 9-16.

Figure 9-16. Translate

For prototyping data engineering workflows, there is no quicker method than serverless technology like Google Cloud Functions. My recommendation is to solve an initial data engineering workflow using serverless technology and move to more complex tools if necessary.

An important note is that the Vertex AI platform adds many additional data engineering and ML engineering components that enhance larger projects. In particular, the ability to use Explainable AI, track the quality of a model, and use a Feature Store are valuable components of a comprehensive MLOps solution. Let's dive into these options next.

Operationalizing ML Models

Every major cloud platform nows has an MLOps platform. On GCP, the platform is Vertex AI and integrates many individual services it has developed over the years, including AutoML technology. In particular, some of the essential components of MLOps platforms include Feature Stores, Explainable AI, and tracking model quality. If starting an MLOps project at a larger company, the first place to start on GCP would be its Vertex AI platform, just like SageMaker on AWS or Azure ML Studio on Azure.

Another option is to use components as standalone solutions to operationalize ML models on the GCP platform. One service to use is the prediction service to deploy models and then accept requests.

For example, you could test a local sklearn model using a command similar to the following:

```
gcloud ai-platform local predict --model-dir\
  LOCAL_OR_CLOUD_STORAGE_PATH_TO_MODEL_DIRECTORY/ \
--json-instances LOCAL_PATH_TO_PREDICTION_INPUT.JSON \
--framework NAME_OF_FRAMEWORK
```

Later, you could create an endpoint and then call this endpoint from an example shown earlier in the chapter, say Google Cloud Functions, Google Cloud Run, or Google App Engine.

Let's walk through an example of how a Google App Engine project looks on the GCP cloud using this repo as a starting point. To start with, notice the core architecture of continuous delivery on GCP. To get started, create a new Google App Engine project as shown in the "light" MLOps workflow in Figure 9-17.

NOTE

Notice that this lightweight workflow allows for a transparent and straightforward way to deploy an ML model, but a "heavy" flow could add tremendous value if the features shown, like Explainable AI, are required for a project.

Figure 9-17. MLOps light versus heavy workflows

Next, enable the Cloud Build API as shown in Figure 9-18.

Figure 9-18. Cloud build

The *cloudbuild.yml* file needs only a deploy command:

```
steps:
- name: "gcr.io/cloud-builders/gcloud"
  args: ["app", "deploy"]
timeout: "1600s"
```

The only other requirements are *app.yaml*, *requirements.txt*, and *main.py*, all found in this example repository. A final step to make this application do any form of machine learning is to call an ML/AI API or use the AI Platform endpoint hosting.

The advantage of the simple approach is that it is easy to set up an entire MLOps pipeline in an hour or two at most. You could also pick and choose from AI APIs, prediction services, and an AutoML endpoint.

There are both "light" and "heavy" approaches to doing MLOps on GCP. This example explored the "light" approach, but there is merit to using the platform technology Vertex AI since it includes advanced features many enterprises desire.

Let's wrap up the chapter and discuss the next steps for using the GCP for MLOps.

Conclusion

One final takeaway on MLOps technologies like Vertex AI is they solve a complicated problem better than most organizations can do themselves. I remember talking to someone at a research lab, and they bragged about how the cloud was overrated since they had a ton of GPUs. A ton of GPUs doesn't give you platform services like these platforms. This statement is a fundamental misunderstanding of how enterprise software and startups work. Comparative advantage is critical for both early-stage startups and Fortune 500 companies. Don't build something worse than what you can buy for a trivial cost.

I recommend taking everything from this chapter and applying it to a final machine learning project that consists of building a cloud native ML application on the GCP. This project should give you the ability to create realistic, working solutions designed with modern techniques.

Before you begin, make sure you read the Sculley et al. (2015) paper to consider technical debt in machine-learning (ML) systems. Your project may benefit from using public data from Google BigQuery datasets. Alternately, if using AutoML, data can be tutorial data or custom data.

The main idea is to create a portfolio project demonstrating your ability to do ML engineering on the Google Cloud. Here are the suggested project requirements to think through:

- Source code stored in GitHub
- Continuous deployment from CircleCI
- Data stored in GCP (BigQuery, Google Cloud Storage, etc.)
- ML predictions created and served out (AutoML, BigQuery, AI Platform, etc.)
- Cloud native monitoring
- Google App Engine serves out HTTP requests via REST API with a JSON payload
- Deployed into GCP environment using Google Cloud Build

Here are some items to add to a final project requirements checklist:

- Does the application make ML inference?
- Are there separate environments?
- Is there comprehensive monitoring and alerts?
- Is the correct datastore used?
- Does the principle of least security apply?
- Is data encrypted in transit?

You can see some recent student and top data science projects in the official code repository. These projects provide a frame of reference for what you can build, and you are welcome to submit a pull request in the future to get your project added.

- Jason Adams: FastAPI Sentiment Analysis with Kubernetes
- James Salafatinos: Tensorflow.js real-time image classification
- Nikhil Bhargava: Sneaker Price Predict
- Covid Predictor
- Absenteeism at Work

The next chapter discusses machine learning interoperability and how it uniquely solves MLOps problems that crop up from different platforms, technologies, and model formats.

Exercises

- Using Google Cloud Shell Editor, create a new GitHub repository with necessary Python scaffolding using a Makefile, linting, and testing. Add steps such as code formatting in your Makefile.
- Create a “hello world” pipeline to Google Cloud that calls into a Python-based Google App Engine (GAE) project and returns “hello world” as a JavaScript Object Notation (JSON) response.
- Create an ingest to ETL pipeline using CSV files and Google BigQuery. Schedule a reoccurring cron job to batch update the data.
- Train a multiclass classification model on Google AutoML vision and deploy to an edge device.
- Create a production and development environment and deploy a project to both environments using Google Cloud Build.

Critical Thinking Discussion Questions

- What problems does a CI system solve, and why is a CI system an essential part of SaaS software?
- Why are cloud platforms the ideal target for analytics applications, and how does deep learning benefit from the cloud?
- What are the advantages of managed services like Google BigQuery, and how does Google BigQuery differ from a traditional SQL?
- How does ML prediction directly from BigQuery add value to the Google platform, and what advantages could this have for analytics application engineering?
- How does AutoML have a lower total cost of ownership (TCO), and how could it have a higher TCO?