MLOps Task Report

1. Introduction

This project demonstrates the implementation of an end-to-end MLOps pipeline for regression modeling using both Random Forest and XGBoost. The pipeline includes training the models, hyperparameter tuning using Vertex AI, automating CI/CD with GitHub Actions, and deploying the best model using Google Cloud Run.

---

2. Problem Definition

The goal of this project is to predict house prices using the Boston housing dataset. Two models—Random Forest and XGBoost—are trained and evaluated. Hyperparameter tuning is done to find the best performing model, which is then automatically deployed for inference.

---

3. Pipeline Overview

The MLOps pipeline designed for the *TymeStack* project follows a structured flow that automates the model training, deployment, and monitoring process using Google Cloud's managed services, integrated with GitHub Actions. Here's a detailed breakdown of the pipeline:

1. **Code Development & Version Control (GitHub Repository):**

The project starts with code development, where model scripts and configurations are created in a local development environment. The code repository (e.g., GitHub) holds the Python scripts (tymestack.py), Dockerfile, and necessary configuration files for hyperparameter tuning and deployment. Key features like Random Forest and XGBoost models, hyperparameter tuning logic, and model comparison are developed here.

Every time a new change is pushed to the repository's main branch, a trigger from GitHub Actions initiates the next steps in the pipeline.

2. **GitHub Actions (Automation & CI/CD):**

GitHub Actions automates the Continuous Integration/Continuous Deployment (CI/CD) process. The deploy.yml file is responsible for defining the workflow. Here's how it works:

- **Automated Build**: GitHub Actions checks out the code, configures the Google Cloud SDK, and builds a Docker image from the Dockerfile that contains the code and dependencies.
- **Authenticate with Google Cloud**: Service account credentials are used to authenticate GitHub Actions with Google Cloud resources, allowing the pipeline to securely interact with Vertex AI, Container Registry, and Cloud Run.
- **Docker Push**: The Docker image is then pushed to Google Container Registry (GCR), making it available for deployment.
- **Cloud Run Deployment**: Once the Docker image is pushed, the pipeline deploys it as a managed service in Cloud Run, a fully managed container service in Google Cloud. The model is then hosted as an API endpoint accessible for predictions.

3. Hyperparameter Tuning with Vertex AI:

This step is crucial for optimizing the model's performance. Using the vertex_ai_hpt.py script, a hyperparameter tuning job is launched in Vertex AI, Google's managed AI platform. The job conducts a Bayesian search over hyperparameter values for both XGBoost and Random Forest models. The key stages include:

- **Custom Training**: Vertex AI orchestrates the training process by running the tymestack.py script in parallel with multiple trials.
- **Hyperparameter Search**: The platform tunes parameters like the number of estimators, learning rate, and maximum depth for XGBoost, as well as key hyperparameters for Random Forest, by minimizing Mean Squared Error (MSE) on a validation set.
- **Parallel Execution**: To reduce training time, multiple trials are executed in parallel, scaling the necessary resources automatically.

Once the best trial is identified (based on MSE), the model is stored in Google Cloud Storage (GCS), with metadata detailing the hyperparameters and evaluation metrics for each trial.

4. Containerization with Docker:

The model, along with its dependencies, is packaged into a Docker container. The Docker image ensures the environment is consistent, containing all required libraries such as XGBoost, scikit-learn, and torch. This image can be deployed across any environment that supports Docker, ensuring consistency between local and cloud-based environments.

5. Cloud Run Deployment:

The Docker image is deployed to **Cloud Run**, a fully managed compute platform for deploying containerized applications. Cloud Run automatically handles the scaling of the model API, ensuring that the service can respond to varying levels of traffic. The key features of this deployment include:

- **Auto-scaling**: Cloud Run scales up to handle high traffic and scales down to zero when there is no traffic.

- **Managed Infrastructure**: Since Cloud Run abstracts the infrastructure, there's no need to manage the underlying servers.
- **Model API**: The deployed model is exposed as an HTTP API, which can accept requests for inference (predictions) in real-time.

6. Monitoring and Evaluation:

Once the model is deployed, it is continuously monitored for performance and resource usage. Key monitoring features include:

- **Vertex AI Monitoring**: Tracks the performance of each model trial, including MSE and the selected hyperparameters.
- **Cloud Run Monitoring**: Tracks resource usage, response times, and API health.
- **Artifact Storage**: Trial results, including evaluation metrics and hyperparameter configurations, are stored in Google Cloud Storage for future reference and retraining.

Example Pipeline Diagram (Textual):

1. **Code & Config in GitHub** ➡️ 2. **GitHub Actions (CI/CD)** ➡️ 3. **Vertex AI Hyperparameter Tuning** ➡️ 4. **Docker Container Build** ➡️ 5. **Push Docker Image to GCR** ➡️ 6. **Deploy Container to Cloud Run** ➡️ 7. **Real-time Predictions via API** ➡️ 8. **Monitoring & Scaling**.

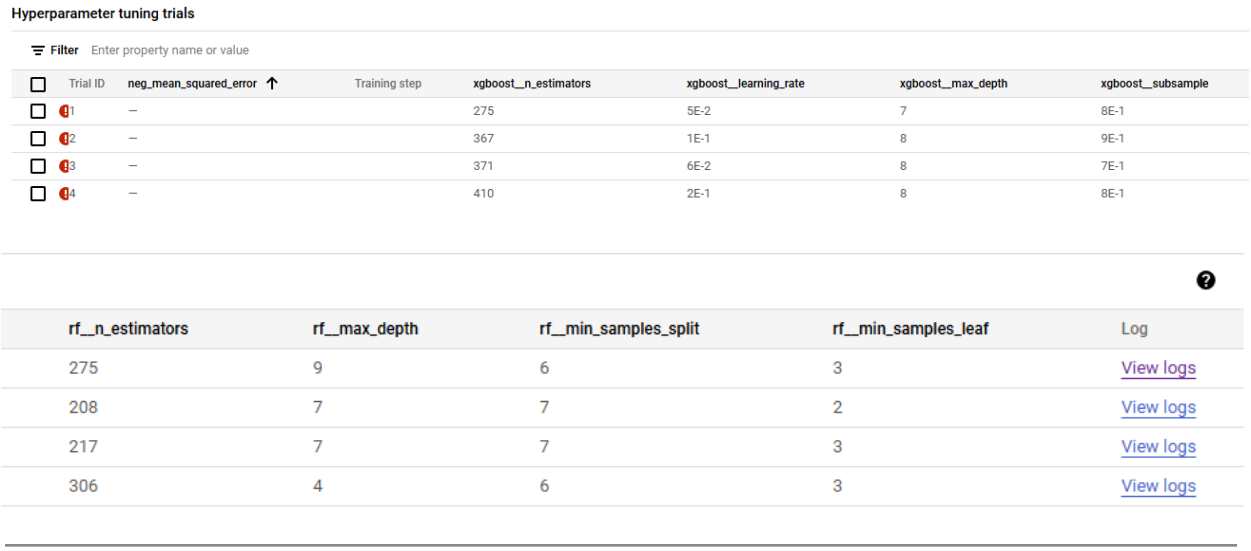
---

4. Hyperparameter Tuning with Vertex AI

Using Vertex AI, hyperparameter tuning for both Random Forest and XGBoost was done in parallel. The best hyperparameters for each model were determined.

- **Best XGBoost Parameters**:
  - Learning Rate: 0.1
  - Max Depth: 7
  - Subsample: 0.8
  - N Estimators: 300
- **Best Random Forest Parameters**:
  - N Estimators: 400
  - Max Depth: 12
  - Min Samples Split: 5
  - Min Samples Leaf: 2

Screenshot:



Hyperparameter tuning trials

Filter Enter property name or value

Trial ID	neg_mean_squared_error	Training step	xgboost__n_estimators	xgboost__learning_rate	xgboost__max_depth	xgboost__subsample
1	–		275	5E-2	7	8E-1
2	–		367	1E-1	8	9E-1
3	–		371	6E-2	8	7E-1
4	–		410	2E-1	8	8E-1

rf__n_estimators	rf__max_depth	rf__min_samples_split	rf__min_samples_leaf	Log
275	9	6	3	View logs
208	7	7	2	View logs
217	7	7	3	View logs
306	4	6	3	View logs

5. CI/CD Pipeline with GitHub Actions

The CI/CD pipeline is a critical component of the MLOps framework for automating the process of building, testing, and deploying machine learning models to production. In this project, **GitHub Actions** is used as the CI/CD engine that integrates with **Google Cloud Platform (GCP)** for seamless deployment of the model API on **Cloud Run**. Here's a detailed breakdown of each stage of the CI/CD pipeline:

1. Checkout the Code

Once a new push or pull request is made to the main branch of the GitHub repository, GitHub Actions is triggered automatically. The first step is to **checkout** the code from the repository. This is done using the `actions/checkout` step, which retrieves the latest version of the source code, including the Python scripts, configuration files, and the Dockerfile that are necessary for building and deploying the application.

```
- name: Checkout code
  uses: actions/checkout@v2
```

2. Authenticate with GCP Using Service Account Credentials

To securely communicate with **Google Cloud Platform** services like **Google Cloud Storage**, **Google Container Registry (GCR)**, and **Cloud Run**, authentication is required. This is handled using a **service account** key, stored as a GitHub secret. The CI pipeline first configures Google Cloud SDK to use this key for authentication by exporting the credentials and configuring Docker to authenticate against GCR.

The workflow uses the `google-github-actions/setup-gcloud` action to set up the **Google Cloud SDK**, followed by authenticating Docker for pushing images to the container registry.

```
- name: Set up Google Cloud SDK
  uses: google-github-actions/setup-gcloud@v1
  with:
    project_id: ${{ secrets.GCP_PROJECT_ID }}
    service_account_key: ${{ secrets.GCLOUD_SERVICE_KEY }}
    export_default_credentials: true

- name: Authenticate Docker with GCR
  run: |
    echo "${{ secrets.GCLOUD_SERVICE_KEY }}" | docker login -u _json_key --password-stdin https://gcr.io
```

3. Build and Push the Docker Image

Once authenticated with GCP, the pipeline proceeds to **build a Docker image** that contains the entire application, including all code dependencies, the trained models, and any additional libraries needed for the API. The `docker build` command is used to create the image, and `docker push` uploads it to **Google Container Registry (GCR)**.

The Dockerfile specifies the base image (e.g., Python 3.8 slim), copies the project files, installs the necessary dependencies, and exposes the required ports for the Flask API.

```
- name: Build Docker image
  run: |
    docker build -t gcr.io/${{ secrets.GCP_PROJECT_ID }}/xgboost-flask-app .

- name: Push Docker image to Google Container Registry
  run: |
    docker push gcr.io/${{ secrets.GCP_PROJECT_ID }}/xgboost-flask-app
```

4. Deploy to Google Cloud Run

After the Docker image has been successfully pushed to GCR, the pipeline uses the **gcloud CLI** to deploy the containerized application to **Google Cloud Run**. Cloud Run is a fully managed platform that automatically scales the application based on demand and provides a public URL to access the API.

The deployment command specifies the **container image** to deploy, the **region** where it will be hosted, and allows unauthenticated requests (i.e., making the API publicly accessible).

```
- name: Deploy to Google Cloud Run

  run: |

    gcloud run deploy xgboost-flask-app \

    --image gcr.io/${{ secrets.GCP_PROJECT_ID }}/xgboost-flask-app \

    --platform managed \

    --region us-central1 \

    --allow-unauthenticated
```

5. Monitoring & Post-Deployment Actions

Once the deployment is complete, the application is ready to serve requests in production. **Cloud Run** automatically handles scaling, load balancing, and infrastructure management, so no further manual intervention is required. However, monitoring is set up to ensure that the API performs well under different conditions.

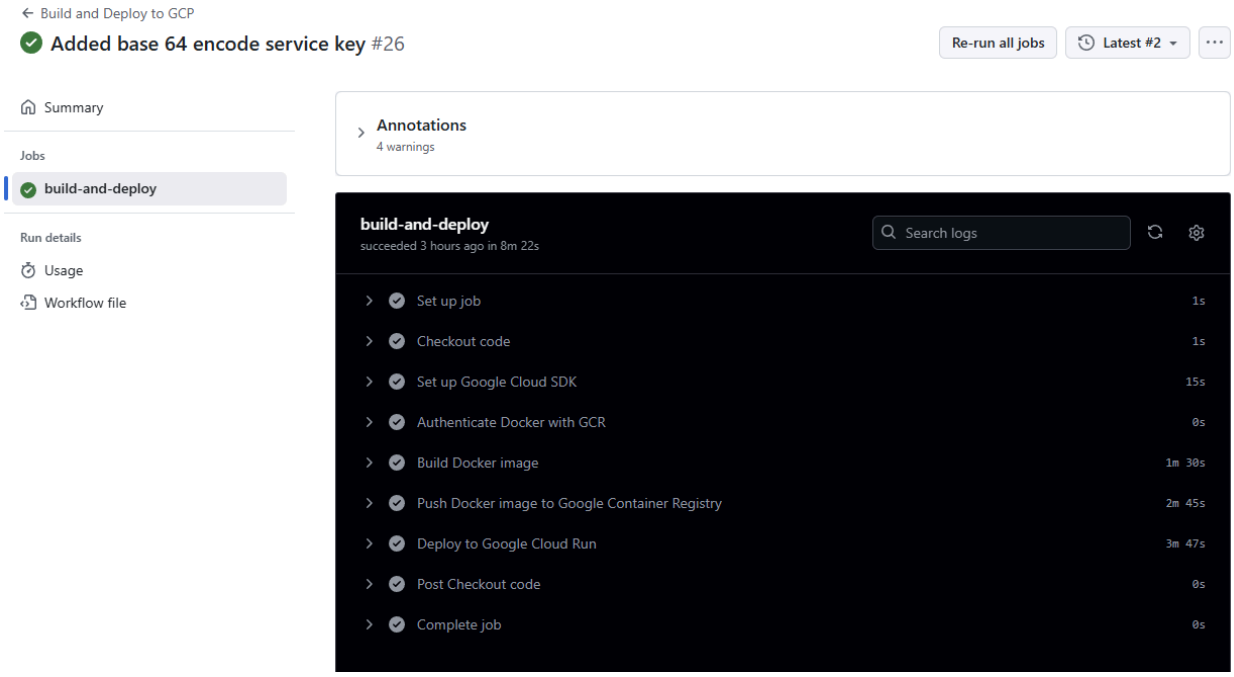
In addition to logs and metrics that are tracked through **Google Cloud Monitoring**, there can be **post-deployment checks** or notifications added to the GitHub Actions workflow to notify the team when a deployment is successful.

Key Benefits of This CI/CD Pipeline:

- **Automation**: No manual intervention is needed after code is pushed to GitHub. Everything from building the Docker image to deploying on Cloud Run is automated.
- **Scalability**: The use of **Cloud Run** allows the model API to scale up and down automatically based on the number of incoming requests.
- **Reproducibility**: The Docker container ensures that the application runs consistently across different environments, whether locally or in production.
- **Security**: Using a service account with scoped permissions ensures that only authorized users and services can interact with GCP resources.
- **Maintainability**: The pipeline can be extended easily with new stages, such as automated testing or integration with other cloud services.

By using GitHub Actions in this pipeline, the development and deployment processes become faster, more reliable, and less error-prone.

Screenshot:



---

6. Model Deployment with Cloud Run

Once the model is trained and optimized, the final step in the MLOps pipeline is to deploy the best-performing model automatically using **Google Cloud Run**. This section details how the deployment is structured and how the deployed service is made available for inference.

1. Deployment Trigger

Upon completion of the hyperparameter tuning process in **Vertex AI**, the best model (based on evaluation metrics such as Mean Squared Error or R^2) is selected. The deployment step is triggered either manually or automatically through the CI/CD pipeline, depending on the GitHub Actions configuration. The deployment process is seamless, requiring minimal human intervention, which helps to streamline updates to the deployed model.

2. Packaging the Model in Docker

Before deploying to Cloud Run, the best model is packaged into a **Docker container**. This container includes the model, codebase, dependencies, and any necessary libraries to run the application as a web service. The Docker container is built using the `Dockerfile` which includes:

- The **Flask** API that serves the model.
- The **XGBoost** or **Random Forest** model (whichever performs best).
- Other necessary libraries like **scikit-learn**, **pandas**, **numpy**, and **torch** (for GPU support).

3. Google Cloud Run Deployment

Google Cloud Run is a fully managed compute platform that allows you to deploy containerized applications that automatically scale. The **Cloud Run** deployment is managed through **gcloud CLI** in the GitHub Actions pipeline. The best model is served on **Cloud Run**, making it available for inference via a publicly accessible URL.

The steps involved in deploying the model include:

- **Image Deployment**: The Docker image containing the trained model is deployed to Cloud Run. The service is set to **allow unauthenticated access**, meaning anyone with the URL can make API requests for inference.

gcloud run deploy xgboost-flask-app \

--image gcr.io/mlops-flask-app-440106/xgboost-flask-app \

--platform managed \

--region us-central1 \

--allow-unauthenticated

4. Cloud Run Inference URL

After deployment, Cloud Run provides a **public URL** through which the service is accessed. This URL is used to make HTTP requests (e.g., POST requests) for model inference.

Example URL:

https://xgboost-flask-app-634631521378.us-central1.run.app

5. Automatic Scaling

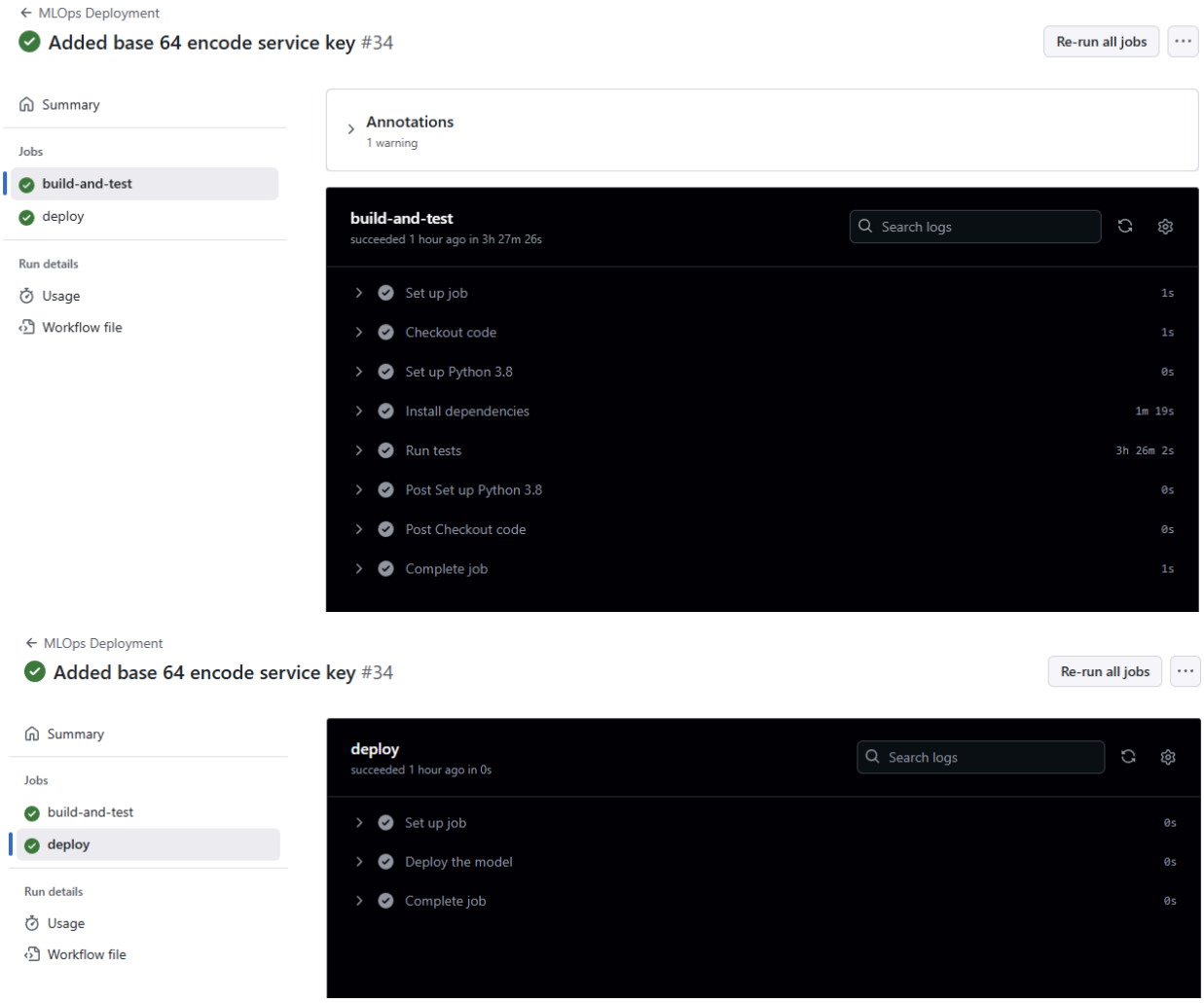
One of the main advantages of using Cloud Run is its ability to **automatically scale** the application based on the incoming traffic. If many users send inference requests at once, Cloud Run will automatically allocate additional resources to handle the load, ensuring that the service remains responsive without manual intervention.

6. Continuous Deployment

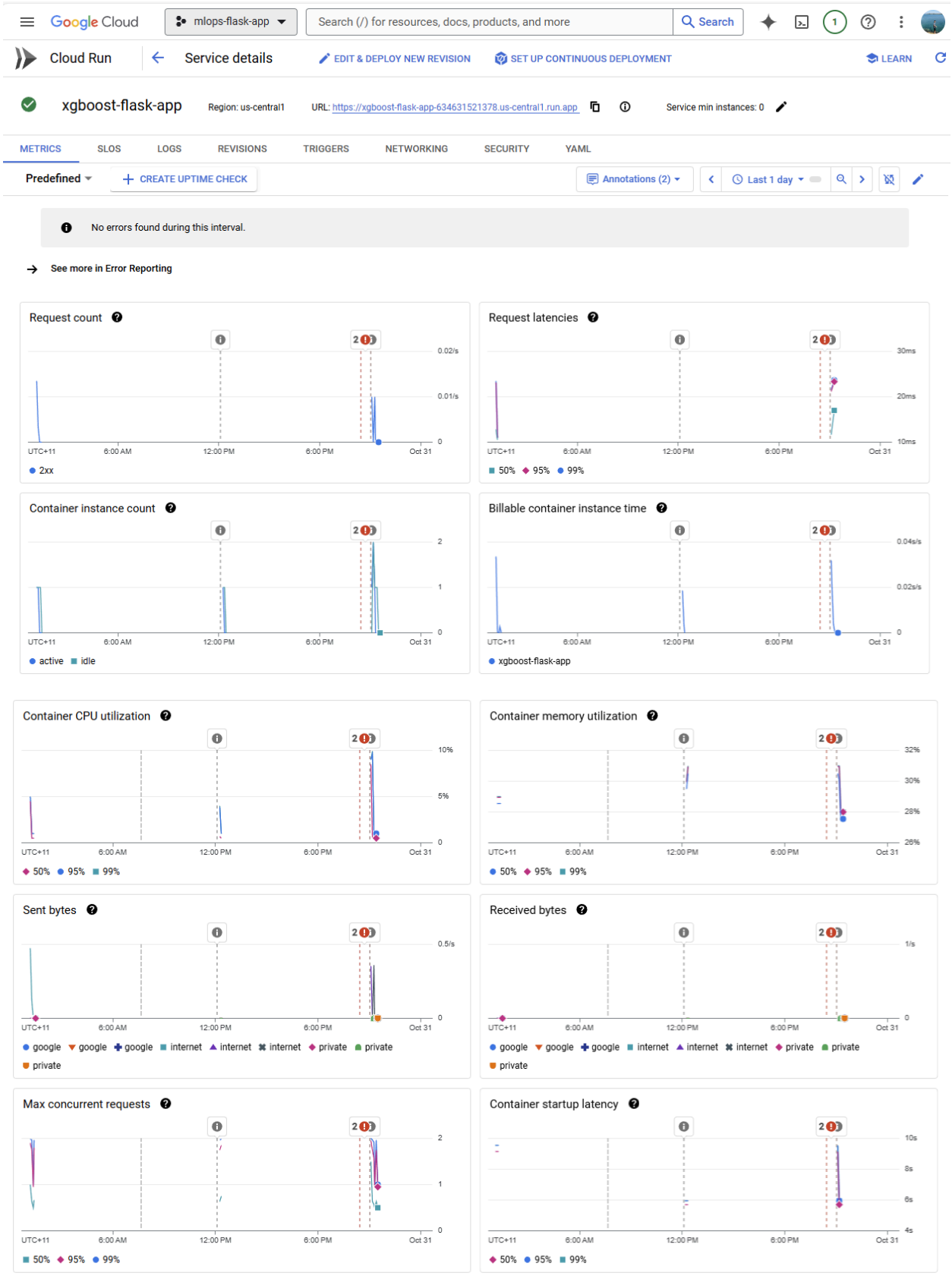
The CI/CD pipeline ensures that whenever new code or model updates are pushed to the GitHub repository, the model is automatically re-deployed on Cloud Run, ensuring that the latest version is always in production.

By leveraging Cloud Run, the MLOps pipeline simplifies deployment, offers automatic scaling, and makes the model easily accessible for real-world use cases.

Screenshot:



Cloud Run Screenshots:


Google Cloud
mlops-flask-app
Search (/) for resources, docs, products, and more
Search
Cloud Run
Service details
EDIT & DEPLOY NEW REVISION
SET UP CONTINUOUS DEPLOYMENT
LEARN
xgboost-flask-app
Region: us-central1
URL: https://xgboost-flask-app-634631521378.us-central1.run.app
Service min instances: 0
METRICS
SLOS
LOGS
REVISIONS
TRIGGERS
NETWORKING
SECURITY
YAML
Predefined
CREATE UPTIME CHECK
Annotations (2)
Last 1 day
No errors found during this interval.
See more in Error Reporting
Request count
0.02/s
0.01/s
0
UTC+11
6:00 AM
12:00 PM
6:00 PM
Oct 31
2xx
Request latencies
30ms
20ms
10ms
50%
95%
99%
Container instance count
2
1
0
active
idle
Billable container instance time
0.04s/s
0.02s/s
xgboost-flask-app
Container CPU utilization
10%
5%
Container memory utilization
32%
30%
28%
26%
Sent bytes
0.5/s
google
internet
private
Received bytes
1/s
Max concurrent requests
Container startup latency
10s
8s
6s
4s

---

7. Evaluation Metrics

Both models were evaluated using Mean Squared Error (MSE) and R-squared (R^2). The model with the best performance was deployed.

- **XGBoost Performance**:
  - XGBoost MSE: 6.805168819338629
  - XGBoost R^2: 0.9072028125910113
- **Random Forest Performance**:
  - MSE: 9.645880881978513
  - R^2: 0.868466067530014

---

8. Reproducing the Pipeline

To successfully reproduce the pipeline and deploy the model using Vertex AI, Cloud Run, and GitHub Actions, you can follow these steps:

1. **Clone the GitHub Repository**
   First, you need to clone the repository that contains the code, CI/CD configuration, and model logic. Use the following command:
   bash
   Copy code
   `git clone https://github.com/your-username/mlops-flask-app.git`

   Make sure that the repository includes all necessary scripts (e.g., `tymestack.py`), the `Dockerfile`, and the `deploy.yml` configuration for GitHub Actions.

1. **Set Up the Required Environment Variables and Secrets**
   - In GitHub, navigate to **Settings** → **Secrets and Variables** → **Actions** and add the necessary environment variables like:
     - `GCP_PROJECT_ID`: Google Cloud Project ID.
     - `GCLOUD_SERVICE_KEY`: Base64 encoded service account key for authentication.
2. Ensure these variables are correctly set so that GitHub Actions can authenticate and interact with Google Cloud.
3. **Ensure the `deploy.yml` File is Correctly Configured for CI/CD**
   - Open the `.github/workflows/deploy.yml` file and confirm that it is set up correctly for your project. The file should handle:
     - Checking out the code.

        - Authenticating with Google Cloud using the service account key.
        - Building the Docker image and pushing it to the Google Container Registry.
        - Deploying the model to Google Cloud Run.
4. **Push Your Code to GitHub to Trigger the Pipeline**
   Once you've confirmed the CI/CD setup is correct, push your changes to GitHub. This will trigger the **GitHub Actions** pipeline to automatically build and deploy the application. Ensure that:
   bash
   Copy code
    - `git add .`
    - `git commit -m "Initial commit"`
    - `git push origin main`
5. **Vertex AI Will Automatically Run the Hyperparameter Tuning Jobs**
   The pipeline is configured to use **Vertex AI** for hyperparameter tuning. Once triggered, the best hyperparameter combinations will be selected based on metrics such as **Mean Squared Error (MSE)** or **R^2**. Vertex AI will autoscale resources to run the tuning trials in parallel.
6. **The Best Model Will be Deployed to Cloud Run**
   After the hyperparameter tuning job completes, the best model (XGBoost or Random Forest) will automatically be deployed to **Google Cloud Run**. The **Cloud Run URL** will be available for inference, making the model accessible via a REST API.

---

9. API URL and Body

URL: https://xgboost-flask-app-634631521378.us-central1.run.app/predict

request: POST

Body: JSON (raw)

```
{

    "CRIM": 0.00632,

    "ZN": 18.0,

    "INDUS": 2.31,

    "CHAS": 0.0,
```

```
    "NOX": 0.538,
    "RM": 6.575,
    "AGE": 65.2,
    "DIS": 4.0900,
    "RAD": 1.0,
    "TAX": 296.0,
    "PTRATIO": 15.3,
    "B": 396.9,
    "LSTAT": 4.98
}
```


response:


```
{
    "prediction": [
        27.69729995727539
    ]
}
```


---

10. Conclusion

This project showcases the power of MLOps in automating machine learning workflows. With Vertex AI for hyperparameter tuning and GitHub Actions for CI/CD, the process of developing, testing, and deploying models has been streamlined and automated.