

# Deployment of a Bank Marketing Prediction Model



AIG 200 - Machine Learning Model Deployment Assignment

Student Name: Johnson YIM

Date: June 22, 2025

## Table of Contents

1	Executive Summary.....	1
2	Machine Learning Problem and Dataset.....	1
3	Model Development Process.....	2
3.1	Exploratory Data Analysis (EDA).....	2
3.2	Data Preprocessing.....	2
3.3	Model Training and Selection.....	3
3.3.1	Comparison of Imbalance Handling Strategies.....	3
3.3.2	Evaluation and Selection of the Core Approach.....	3
3.3.3	Hyperparameter Tuning and Final Model Selection.....	4
3.3.4	Saving the Model Artifacts.....	4
4	Deployment Architecture.....	5
4.1	Tools and Services.....	5
4.2	System Architecture.....	5
4.3	API Endpoint and Security.....	6
5	Deployment Process.....	7
5.1	Local API Development and Containerization.....	7
5.2	Cloud Platform Setup and Deployment.....	8
6	Challenges.....	12
7	Live Deployment Verification.....	15
8	Conclusion and Future Improvements.....	19
9	References.....	20

# 1 Executive Summary

The primary objective of this project was to gain hands-on experience in deploying a machine learning model to a cloud environment. This was achieved by developing a neural network model to predict the success of a bank's term deposit marketing campaign and deploying it as a scalable, secure REST API. The model was trained on the UCI Bank Marketing dataset, with special attention given to handling class imbalance through class weighting. The final application was containerized using Docker and successfully deployed on the Google Cloud Run platform, providing a functional and resilient API endpoint for real-time predictions. The project successfully navigated the entire machine learning lifecycle, from data exploration and preprocessing to a production-ready cloud deployment.

## 2 Machine Learning Problem and Dataset

The project addresses a common business challenge in the banking sector: improving the efficiency of direct marketing campaigns. The goal is to predict whether a client will subscribe to a term deposit based on their demographic data, past interactions, and prevailing economic indicators. By accurately identifying potential subscribers, the bank can optimize its marketing efforts and improve conversion rates.

The dataset used is the Bank Marketing Dataset from the UCI Machine Learning Repository. It contains 41,188 records and 20 input features, including:

- Client Data: age, job, marital status, education.
- Last Contact Data: contact type, month, day of the week.
- Campaign History: number of contacts (campaign), days since last contact (pdays), and outcome of the previous campaign (poutcome).
- Economic Indicators: employment variation rate, consumer confidence index, etc.

The target variable is a binary outcome (y), indicating "yes" or "no" to a term deposit subscription.

Data Source is available at: Moro, S., Rita, P., & Cortez, P. (2014). Bank Marketing [Dataset]. UCI Machine Learning Repository. <https://doi.org/10.24432/C5K306>.

## 3 Model Development Process

The model development process focused on creating a robust pipeline that could be reliably operationalized at inference time.

### 3.1 Exploratory Data Analysis (EDA)

Initial analysis of the dataset revealed several key characteristics:

- **Class Imbalance:** The dataset is highly imbalanced, with approximately 89% of clients not subscribing and only 11% subscribing. This required special handling during training to prevent model bias towards the majority class.
- **Feature Distribution:** Numerical features like *campaign* were heavily right-skewed. High-cardinality categorical features like *job* required an efficient encoding strategy.
- **Data Leakage Potential:** The *duration* feature (call duration) was identified as a source of data leakage, as it is only known *after* a call is completed and a decision is made. Therefore, it was excluded from the model.

### 3.2 Data Preprocessing

A robust preprocessing pipeline was developed to clean and transform the data for the neural network model:

- **Data Cleaning:** Rows with unknown values in key categorical columns (e.g., marital) were dropped. The default column was also removed due to having very few "yes" instances, providing little predictive signal.
- **Encoding:** A mix of encoding strategies was employed:
  - **Target Encoding:** Used for high-cardinality features like *job* and *poutcome*.
  - **Ordinal Encoding:** Applied to the *education* feature to preserve its inherent order.
  - **One-Hot & Binary Encoding:** Used for lower-cardinality features like *contact*, *housing*, and *loan*.
- **Feature Engineering:** The *pdays* and *previous* columns were engineered into more informative binary features: *previously\_contacted* and *was\_contacted\_before*.
- **Transformation & Scaling:** The skewed campaign feature was log-transformed. All numerical features were scaled using StandardScaler to normalize their range and ensure stable training.

### 3.3 Model Training and Selection

A multi-stage training process was conducted to identify the best model for deployment. The core architecture was a feedforward neural network, chosen for its ability to capture complex, non-linear patterns in the data. The primary challenge was the significant class imbalance in the dataset (~11% positive class). Therefore, two fundamentally different modeling approaches were trained and compared, both using a neural network but differing in their strategy for handling this imbalance.

#### 3.3.1 Comparison of Imbalance Handling Strategies

A baseline neural network architecture (2 hidden layers, ReLU activation, Dropout) was used to evaluate two distinct training strategies:

- **Model 1: SMOTE (Synthetic Minority Oversampling Technique):** In this approach, the training data was first balanced using SMOTE, which creates synthetic samples for the minority class ("yes" subscribers). The neural network was then trained on this artificially balanced dataset.
- **Model 2: Class Weighting:** This strategy addresses the imbalance directly within the model's training process. A higher penalty (weight) was assigned to errors made on the minority class. This forces the model to pay more attention to correctly classifying potential subscribers without altering the original data distribution.

#### 3.3.2 Evaluation and Selection of the Core Approach

The business objective of the marketing campaign is to maximize outreach to potential subscribers. Therefore, the most critical evaluation metric is **Recall** for the positive class ("yes"), as it measures the model's ability to identify the highest possible percentage of actual subscribers. A model with high recall minimizes the risk of "missed opportunities."

The performance of the two models was evaluated on the test set at a standard 0.5 probability threshold:

Metric	SMOTE Model	Class Weight Model
Recall (Yes)	56%	61%
Precision (Yes)	43%	41%
F1-Score (Yes)	48%	49%
ROC AUC	0.79	0.80

**Selection:** The **Class Weighting model was selected** as the superior approach. It achieved a significantly higher recall (61%) than the SMOTE model (56%), meaning it successfully identified a larger portion of true subscribers. While its precision was slightly lower, the higher recall and slightly better F1-score and AUC made it the better choice for the business objective. Furthermore, the Class Weight model exhibited better generalization with less overfitting compared to the SMOTE model.

### 3.3.3 Hyperparameter Tuning and Final Model Selection

After selecting the Class Weighting strategy, further hyperparameter tuning was conducted to optimize the neural network's architecture. Three configurations were tested, varying in depth, activation function, and regularization:

- **Config A (Baseline):** 2 hidden layers ([32, 16]), relu activation, 0.2 dropout.
- **Config B (Moderate Depth):** 3 hidden layers ([64, 32, 16]), relu activation, 0.3 dropout.
- **Config C (Deep):** 3 hidden layers ([128, 64, 32]), tanh activation, rmsprop optimizer.

To ensure a fair comparison based on the business goal, the models were evaluated at an **optimized threshold of 0.58**, which was determined via threshold sweeping to maximize the F1-score for the Class Weight strategy.

Metric	Config A	Config B	Config C
Accuracy	85%	87%	<b>88%</b>
Precision (Yes)	38%	44%	<b>47%</b>
Recall (Yes)	<b>61%</b>	60%	49%
F1-Score (Yes)	48%	<b>51%</b>	48%
ROC AUC	<b>80%</b>	<b>80%</b>	78%

**Final Selection:** While Config B had the highest F1-Score, **Config A was selected as the final model for deployment**. This decision was driven by the primary business objective: Config A provided the highest **Recall (61%)** and a top-tier **ROC AUC (80%)**. In a marketing context, the higher cost of a missed opportunity (a false negative) outweighs the cost of contacting a non-subscriber (a false positive). Therefore, the model that captures the most potential subscribers is the most preferable.

### 3.3.4 Saving the Model Artifacts

The final trained model (Config A) was saved as a *.keras* file. The corresponding preprocessing pipeline, including the fitted *TargetEncoder* and *StandardScaler* objects, was saved using *joblib*. These artifacts are essential for ensuring that new data at inference time is processed in the exact same way as the training data.

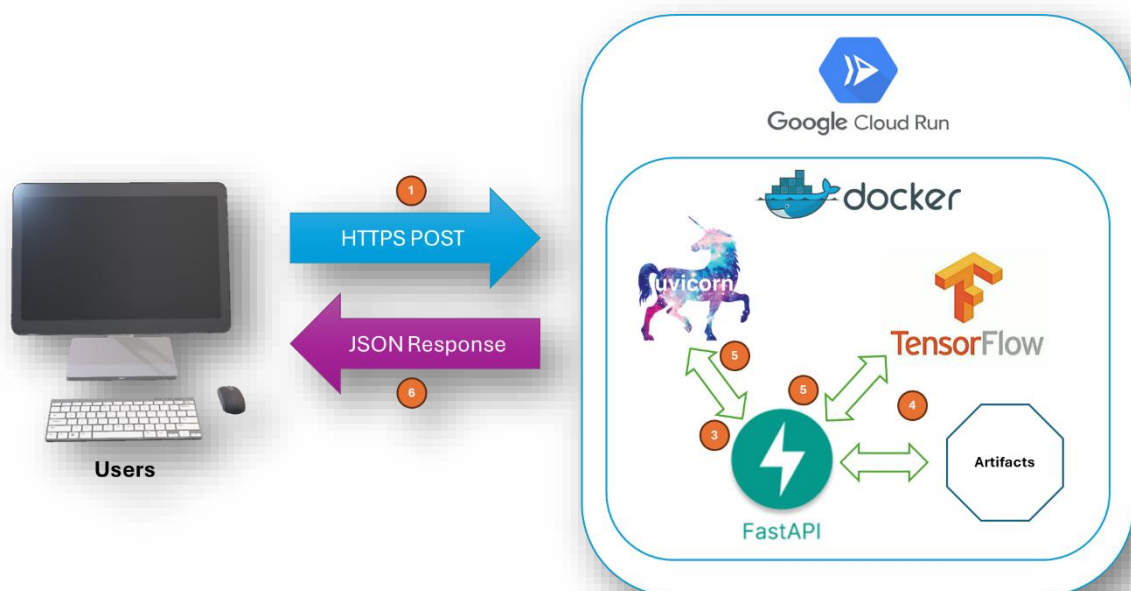
## 4 Deployment Architecture

The trained model was deployed on **Google Cloud Platform** and accessible via a REST API.

### 4.1 Tools and Services

- **Web Framework: FastAPI** was chosen for its high performance and automatic generation of interactive API documentation (Swagger UI).
- **Containerization: Docker** was used to package the application, model artifacts, and dependencies into a portable and reproducible container.
- **Cloud Platform: Google Cloud Platform (GCP)** was selected, specifically using **Cloud Run** for deployment. Cloud Run is a fully managed, serverless platform that automatically scales instances, fitting the scalable and resilient requirements.

### 4.2 System Architecture



System Architecture Diagram

The system architecture follows a simple, robust pattern for serverless ML deployment:

1. A **User or Client Application** sends an HTTPS POST request with the client's data in JSON format to the public API endpoint.
2. The request hits the **Google Cloud Run** service, which automatically provisions and runs an instance of our application container if one isn't already running.
3. Inside the **Docker Container**, the Uvicorn server passes the request to the **FastAPI Application**.

4. The API validates the API key, preprocesses the incoming data using the loaded artifacts, and feeds it to the loaded **TensorFlow/Keras Model**.
5. The model returns a prediction probability, which is converted to a "yes" or "no" label based on the optimized threshold.
6. The final **JSON Response** is sent back through the stack to the user.

### 4.3 API Endpoint and Security

- **Endpoint:** A /predict endpoint was created that accepts POST requests containing client data in JSON format.
- **Security:** Basic API key authentication was implemented as required. The API checks for a custom header, *x-api-key*, and rejects any requests that do not contain the valid key, ensuring a basic level of protection for the endpoint.



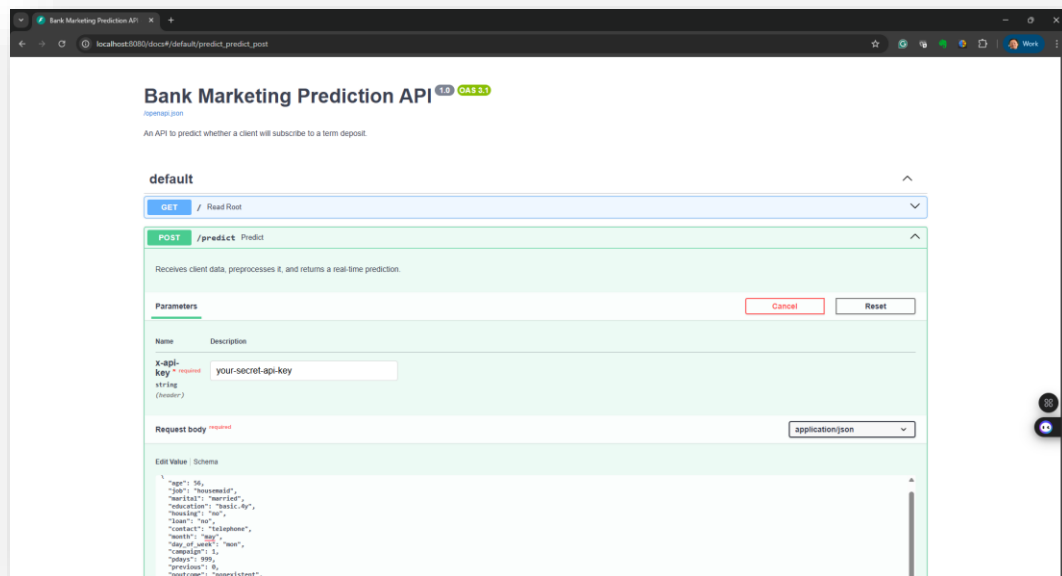
## 5 Deployment Process

The deployment process followed a structured, iterative path to containerize the application and launch it on a scalable cloud platform.

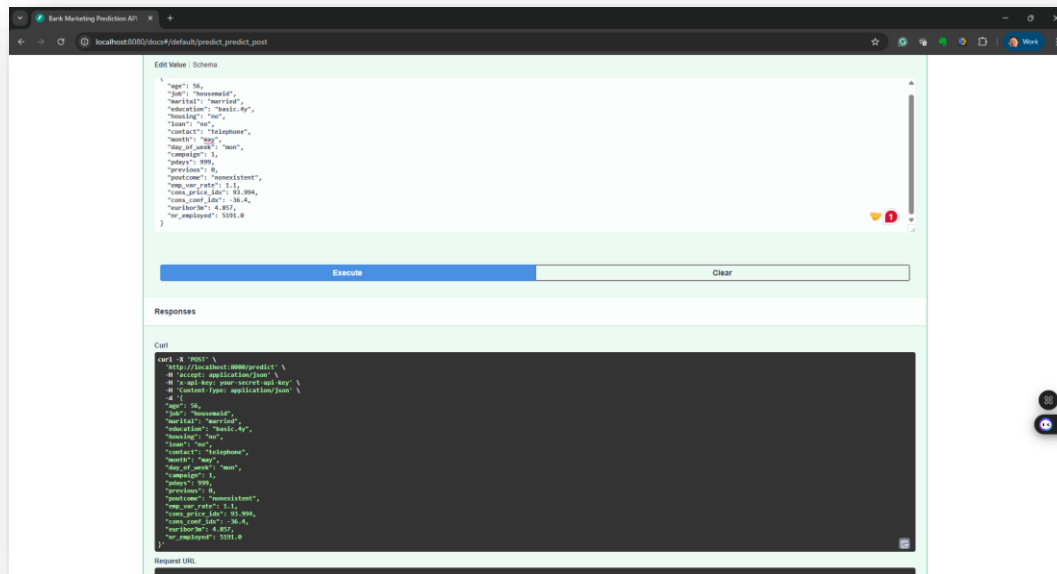
### 5.1 Local API Development and Containerization

The first step was to create a web application to serve the model's predictions.

- **API Framework: FastAPI** was chosen to build the application due to its high performance and automatic generation of interactive API documentation, which is invaluable for testing. The application code was written in `main.py`.
- **Dependency Management:** A `requirements.txt` file was created to explicitly list all necessary Python libraries (e.g., `tensorflow`, `pandas`, `fastapi`, `uvicorn`), ensuring a reproducible environment.
- **Containerization:** A **Dockerfile** was written to define the steps for building a portable container image. This file specified the base Python image, copied all project files (including the saved model and preprocessors), installed the dependencies from `requirements.txt`, and set the command to start the Uvicorn server.
- **Local Verification:** Before deploying to the cloud, the Docker container was built and run locally. The local API endpoint (`http://localhost:8080/docs`) was tested to confirm that the containerized application could successfully load the model and return predictions.



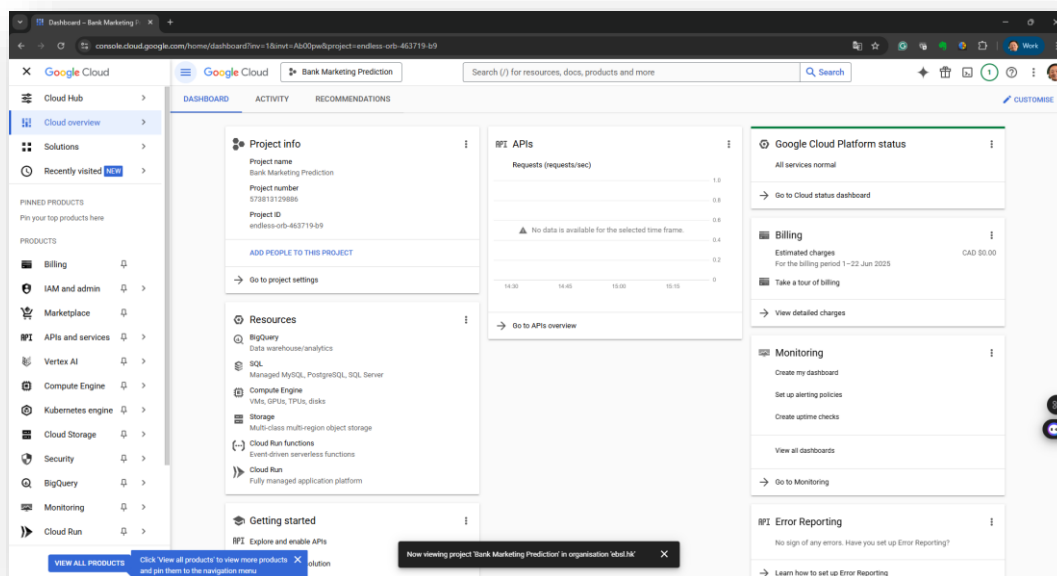
Local Verification with FastAPI Interactive Documentation



Local Testing Response

## 5.2 Cloud Platform Setup and Deployment

- **Platform Selection:** Google Cloud Run was selected as the deployment platform for its fully managed, serverless architecture, which meets the requirement for a **scalable and resilient** solution.



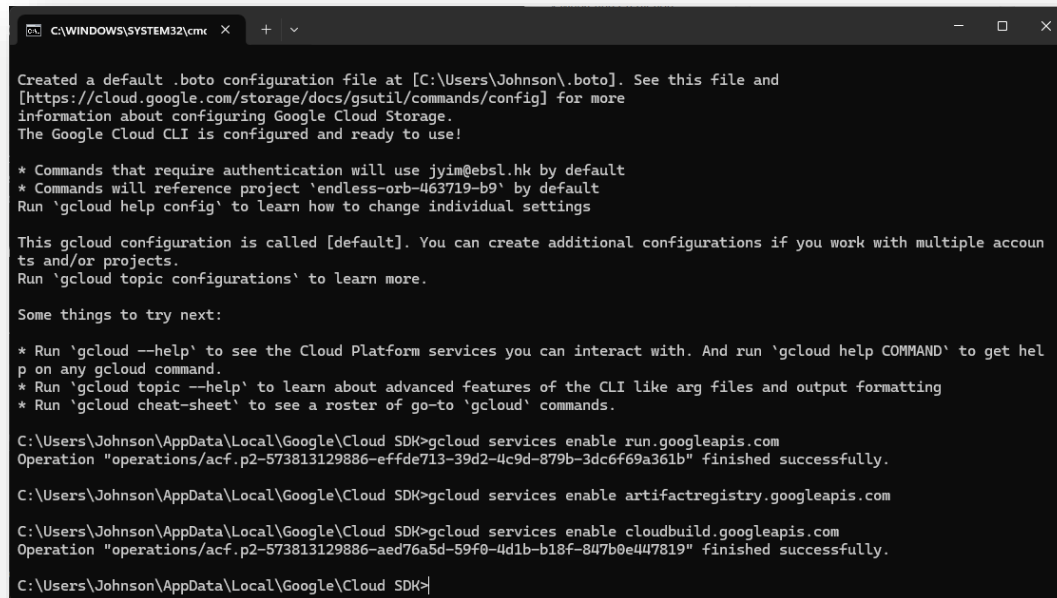
Google Cloud – Bank Marketing Prediction Project Dashboard

The following commands enable the APIs for Cloud Run, Cloud Build, and Artifact Registry.

```
gcloud services enable run.googleapis.com
```

```
gcloud services enable artifactregistry.googleapis.com
```

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



```
C:\WINDOWS\SYSTEM32\cmd X + v

Created a default .boto configuration file at [C:\Users\Johnson\.boto]. See this file and
[https://cloud.google.com/storage/docs/gsutil/commands/config] for more
information about configuring Google Cloud Storage.
The Google Cloud CLI is configured and ready to use!

* Commands that require authentication will use jyim@ebsl.hk by default
* Commands will reference project 'endless-orb-463719-b9' by default
Run 'gcloud help config' to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configurations if you work with multiple account
s and/or projects.
Run 'gcloud topic configurations' to learn more.

Some things to try next:

* Run 'gcloud --help' to see the Cloud Platform services you can interact with. And run 'gcloud help COMMAND' to get hel
p on any gcloud command.
* Run 'gcloud topic --help' to learn about advanced features of the CLI like arg files and output formatting
* Run 'gcloud cheat-sheet' to see a roster of go-to 'gcloud' commands.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable run.googleapis.com
Operation "operations/acf.p2-573813129886-effde713-39d2-4c9d-879b-3dc6f69a361b" finished successfully.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable artifactregistry.googleapis.com

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable cloudbuild.googleapis.com
Operation "operations/acf.p2-573813129886-aed76a5d-59f0-4d1b-b18f-847b0e447819" finished successfully.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>
```

### Enable APIs on Google Cloud Run, Cloud Build and Artifact Registry

- **Cloud Build & Artifact Registry:** The local project code was submitted to **Google Cloud Build**, which automatically built the Docker image in the cloud and pushed it to a private **Artifact Registry** repository.

The following commands instruct Google Cloud to create a private registry to store the Docker image; and Build to take the code from current directory, build the Docker image in the cloud, and push it to the Artifact Registry repository.

```
gcloud artifacts repositories create bank-marketing-repo --repository-format=docker
--location=us-central1
```

```
gcloud builds submit --tag us-central1-docker.pkg.dev/endless-orb-463719-b9/bank-
marketing-repo/bank-marketing-api
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
Run 'gcloud help config' to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configurations if you work with multiple accounts and/or projects.
Run 'gcloud topic configurations' to learn more.

Some things to try next:

* Run 'gcloud --help' to see the Cloud Platform services you can interact with. And run 'gcloud help COMMAND' to get help on any gcloud command.
* Run 'gcloud topic --help' to learn about advanced features of the CLI like arg files and output formatting
* Run 'gcloud cheat-sheet' to see a roster of go-to 'gcloud' commands.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable run.googleapis.com
Operation "operations/acf.p2-573813129886-efde713-39d2-4c9d-879b-3dc6f69a361b" finished successfully.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable artifactregistry.googleapis.com

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud services enable cloudbuild.googleapis.com
Operation "operations/acf.p2-573813129886-aed76a5d-59f0-4d1b-b18f-847b0e447819" finished successfully.

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>gcloud artifacts repositories create bank-marketing-repo --repository-format=docker --location=us-central1
Create request issued for: [bank-marketing-repo]
Waiting for operation [projects/endlless-orb-463719-b9/locations/us-central1/operations/cf749a61-b2c6-45a3-8661-2f16dea9c607] to complete...done.
Created repository [bank-marketing-repo].

C:\Users\Johnson\AppData\Local\Google\Cloud SDK>
```

## Create a Docker Registry

```
C:\WINDOWS\SYSTEM32\cmd.exe
PUSH
Pushing us-central1-docker.pkg.dev/endlless-orb-463719-b9/bank-marketing-repo/bank-marketing-api
The push refers to repository [us-central1-docker.pkg.dev/endlless-orb-463719-b9/bank-marketing-repo/bank-marketing-api]
8281759cdc3b: Preparing
8c1439a9c222: Preparing
114c9ealafa6: Preparing
63e09f79cfb7: Preparing
f3221a8c83dd: Preparing
e4e2acb8cf69: Preparing
7fb72a7d1a8e: Preparing
e4e2acb8cf69: Waiting
7fb72a7d1a8e: Waiting
8c1439a9c222: Pushed
114c9ealafa6: Pushed
63e09f79cfb7: Pushed
e4e2acb8cf69: Pushed
f3221a8c83dd: Pushed
7fb72a7d1a8e: Pushed
8281759cdc3b: Pushed
latest: digest: sha256:5b36d1816350d7852b2511232662afdc015eb227468bcd58236a53df6568a88 size: 1788
DONE

ID: 55de2f1d-fec1-4069-a1fd-a7804cef7f5e
CREATE_TIME: 2025-06-22T20:06:09+00:00
DURATION: 3M45S
SOURCE: gs://endlless-orb-463719-b9_cloudbuild/source/1750622767.424757-0168b06343584e42b89aae79b68174a3.tgz
IMAGES: us-central1-docker.pkg.dev/endlless-orb-463719-b9/bank-marketing-repo/bank-marketing-api (+1 more)
STATUS: SUCCESS

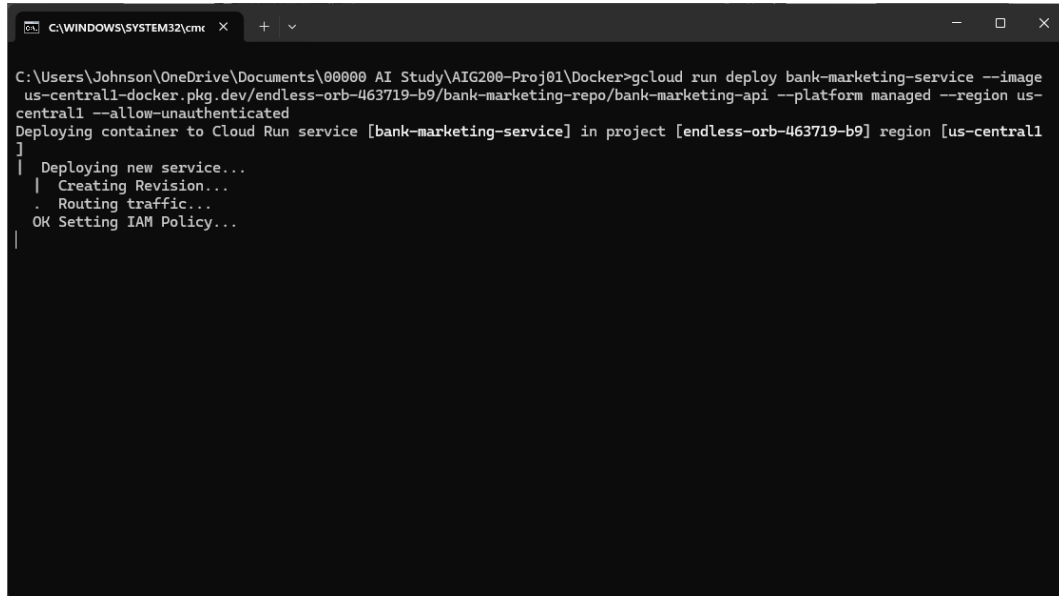
C:\Users\Johnson\OneDrive\Documents\000000 AI Study\AIG200-Proj01\Docker>
```

## Build and Push Completion

- **Initial Deployment:** The *gcloud run deploy* command was used to launch the container image from the Artifact Registry as a new service named bank-marketing-service.

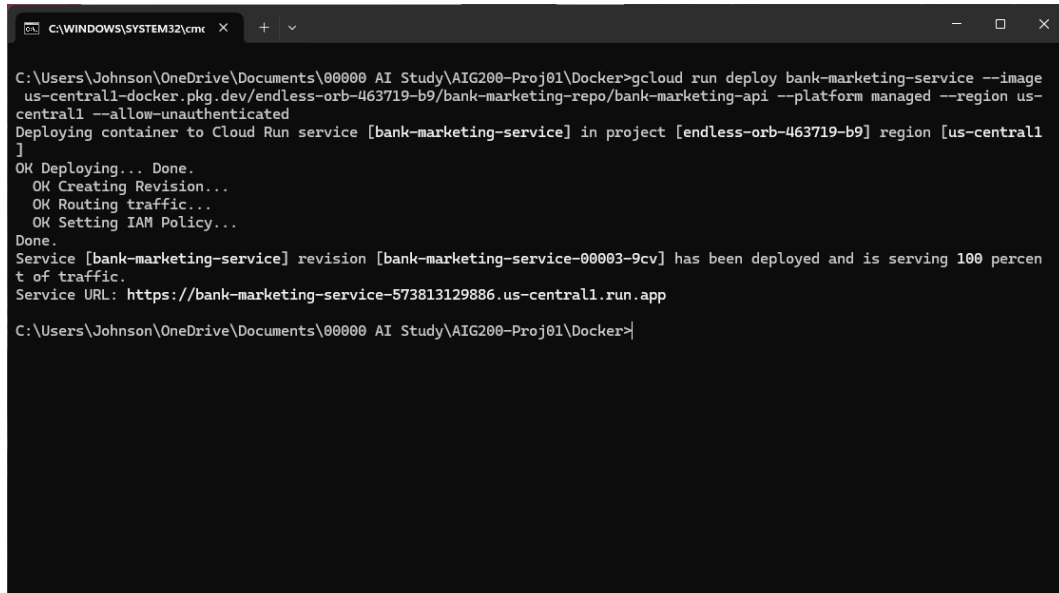
The following command takes the image from the registry and deploy it to Cloud Run as a live web service.

```
gcloud run deploy bank-marketing-service --image us-central1-  
docker.pkg.dev/etl-org-463719-b9/bank-marketing-repo/bank-marketing-api --  
platform managed --region us-central1 --allow-unauthenticated
```



```
C:\WINDOWS\SYSTEM32\cmd X + v  
C:\Users\Johnson\OneDrive\Documents\00000 AI Study\AIG200-Proj01\Docker>gcloud run deploy bank-marketing-service --image  
us-central1-docker.pkg.dev/etl-org-463719-b9/bank-marketing-repo/bank-marketing-api --platform managed --region us-  
central1 --allow-unauthenticated  
Deploying container to Cloud Run service [bank-marketing-service] in project [etl-org-463719-b9] region [us-central1]  
]  
| Deploying new service...  
| Creating Revision...  
| Routing traffic...  
OK Setting IAM Policy...  
|
```

Cloud Run Deployment in Progress



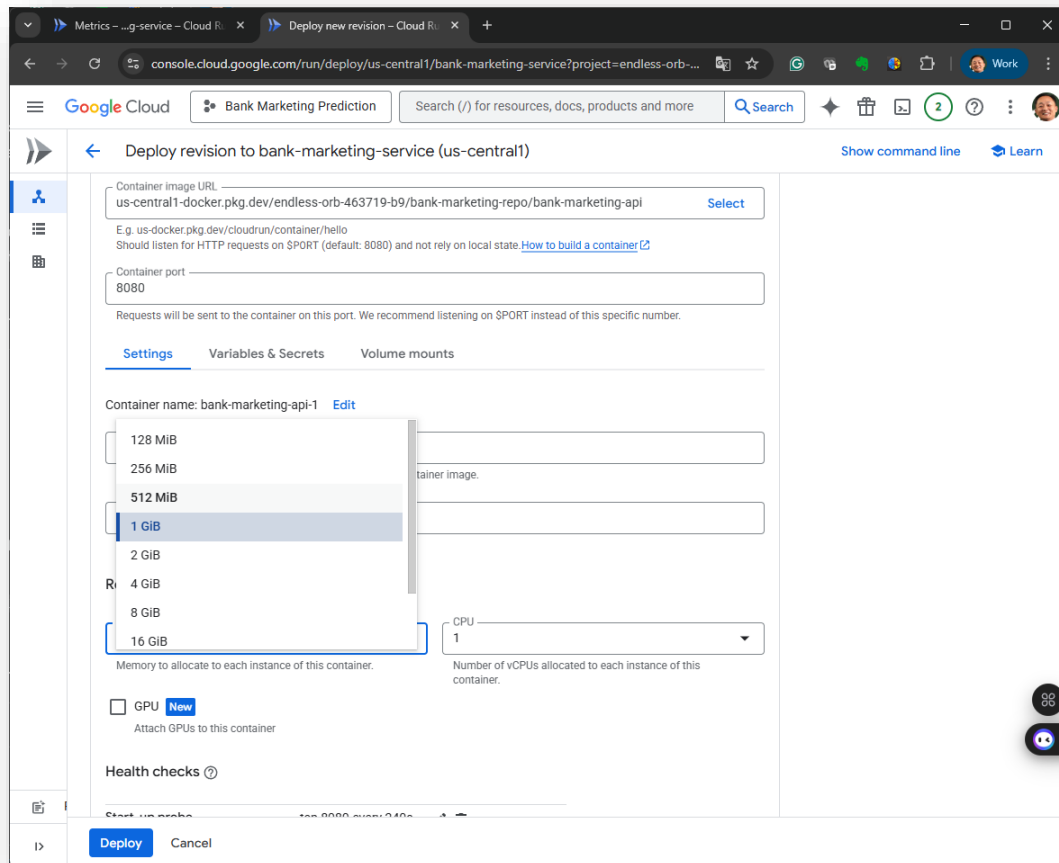
```
C:\WINDOWS\SYSTEM32\cmd X + v  
C:\Users\Johnson\OneDrive\Documents\00000 AI Study\AIG200-Proj01\Docker>gcloud run deploy bank-marketing-service --image  
us-central1-docker.pkg.dev/etl-org-463719-b9/bank-marketing-repo/bank-marketing-api --platform managed --region us-  
central1 --allow-unauthenticated  
Deploying container to Cloud Run service [bank-marketing-service] in project [etl-org-463719-b9] region [us-central1]  
]  
OK Deploying... Done.  
OK Creating Revision...  
OK Routing traffic...  
OK Setting IAM Policy...  
Done.  
Service [bank-marketing-service] revision [bank-marketing-service-00003-9cv] has been deployed and is serving 100 percent  
of traffic.  
Service URL: https://bank-marketing-service-573813129886.us-central1.run.app  
C:\Users\Johnson\OneDrive\Documents\00000 AI Study\AIG200-Proj01\Docker>
```

Cloud Run Deployment Completed

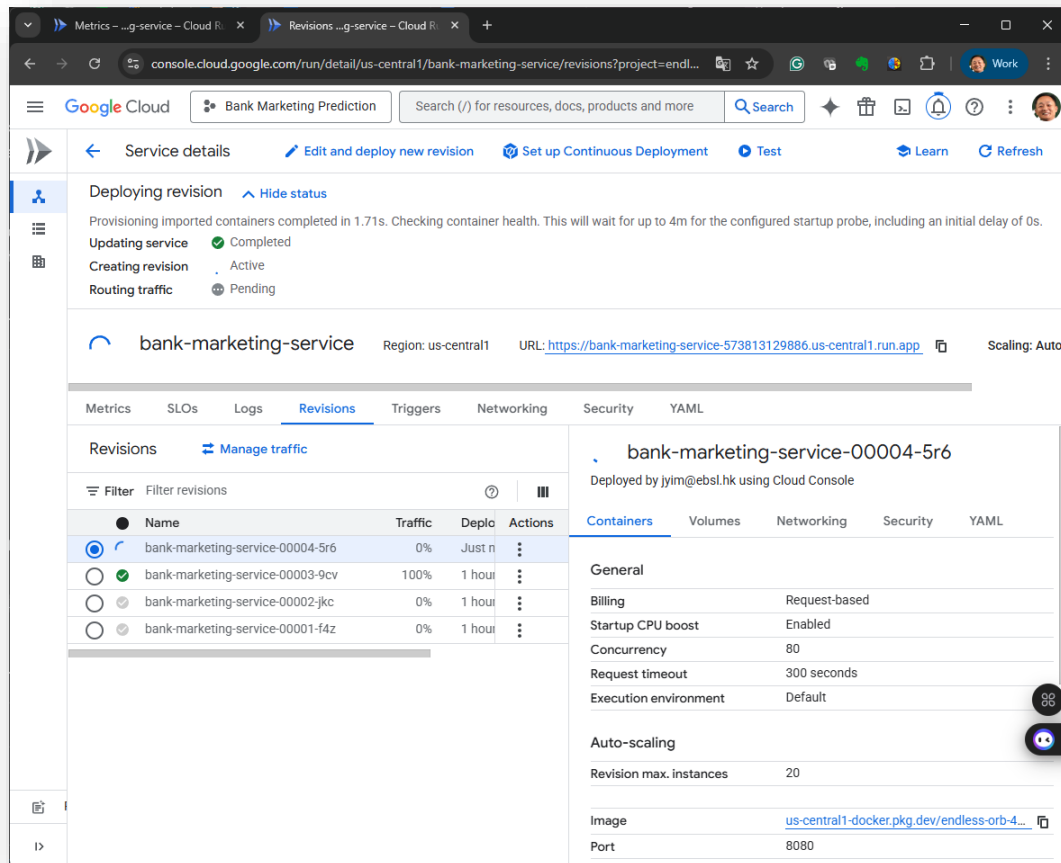
## 6 Challenges

Several practical challenges were encountered and resolved during the deployment process.

- **Dependency Management:** The initial attempt to containerize the application failed with a "uvicorn: executable file not found" error. This was traced back to an incomplete requirements.txt file generated from the local conda environment. The issue was resolved by manually creating a clean requirements.txt file with only the essential libraries and rebuilding the Docker image using the --no-cache flag to ensure a fresh installation.
- **Port Configuration:** The first deployment failed because the container did not start correctly. The error logs indicated that the application was not listening on the port provided by Cloud Run's `PORT` environment variable. This was because the port was hardcoded in the Dockerfile. The issue was resolved by changing the Dockerfile's `CMD` instruction to use the `$PORT` environment variable, making the application compliant with the Cloud Run environment. The image was then rebuilt and redeployed.
- **Cloud Resource Configuration:** The initial deployment to Google Cloud Run encountered system halt during a number of CURL test runs. Investigation with system logs indicating the service crashed because it **exceeded its memory limit**. The default 512 MiB allocation was insufficient for loading the TensorFlow library and the model. This was resolved by redeploying the service with an increased memory allocation of **1 GiB** through the Google Cloud Console, which stabilized the application. Screen captures of the fixes at Google Cloud Console are depicted below.



Upgrade Memory Limit for current Model



### Redeploy Model after Memory Upgrade

This iterative process of deploying, reading logs, and fixing configuration issues is a critical part of the MLOps lifecycle and was a key learning experience in this project.



## 7 Live Deployment Verification

The API was successfully deployed and tested.

---

<b>Live API Endpoint</b>	<a href="https://bank-marketing-service-573813129886.us-central1.run.app/predict">https://bank-marketing-service-573813129886.us-central1.run.app/predict</a>
--------------------------	---

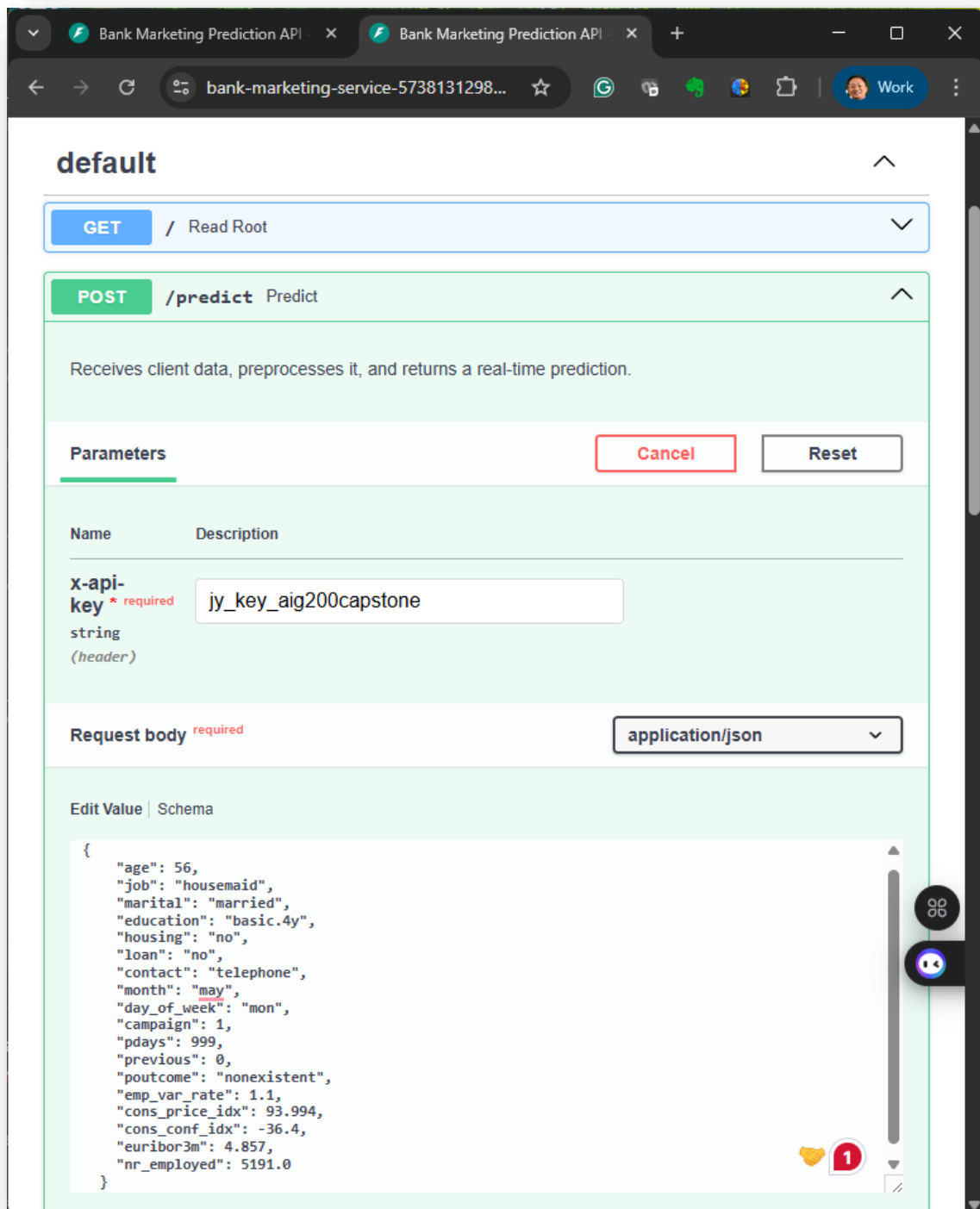
<b>Interactive API Documentation (for browser-based testing):**</b>	<a href="https://bank-marketing-service-573813129886.us-central1.run.app/docs#/default/predict_predict_post">https://bank-marketing-service-573813129886.us-central1.run.app/docs#/default/predict_predict_post</a>
---	---

<b>API Key</b>	jy_key_aig200capstone
----------------	-----------------------

### Testing Data in JSON

```
{
  "age": 56,
  "job": "housemaid",
  "marital": "married",
  "education": "basic.4y",
  "housing": "no",
  "loan": "no",
  "contact": "telephone",
  "month": "may",
  "day_of_week": "mon",
  "campaign": 1,
  "pdays": 999,
  "previous": 0,
  "poutcome": "nonexistent",
  "emp_var_rate": 1.1,
  "cons_price_idx": 93.994,
  "cons_conf_idx": -36.4,
  "euribor3m": 4.857,
  "nr_employed": 5191.0
}
```

The following screenshots show successful test of the live endpoint using the interactive documentation provided by FastAPI.



FastAPI Interactive Documentation Request

Bank Marketing Prediction API

Responses

Curl

```
curl -X 'POST' \
  'https://bank-marketing-service-573813129886.us-central1.run.app/predict' \
  -H 'accept: application/json' \
  -H 'x-api-key: jy_key_aig200capstone' \
  -H 'Content-Type: application/json' \
  -d '{
    "age": 56,
    "job": "housemaid",
    "marital": "married",
    "education": "basic.4y",
    "housing": "no",
    "loan": "no",
    "contact": "telephone",
    "month": "may",
    "day_of_week": "mon",
    "campaign": 1,
    "pdays": 999,
    "previous": 0,
    "poutcome": "nonexistent",
    "emp_var_rate": 1.1,
    "cons_price_idx": 93.994,
    "cons_conf_idx": -36.4,
    "euribor3m": 4.857,
    "nr_employed": 5191.0
  }'
```

Request URL

https://bank-marketing-service-573813129886.us-central1.run.app/predict

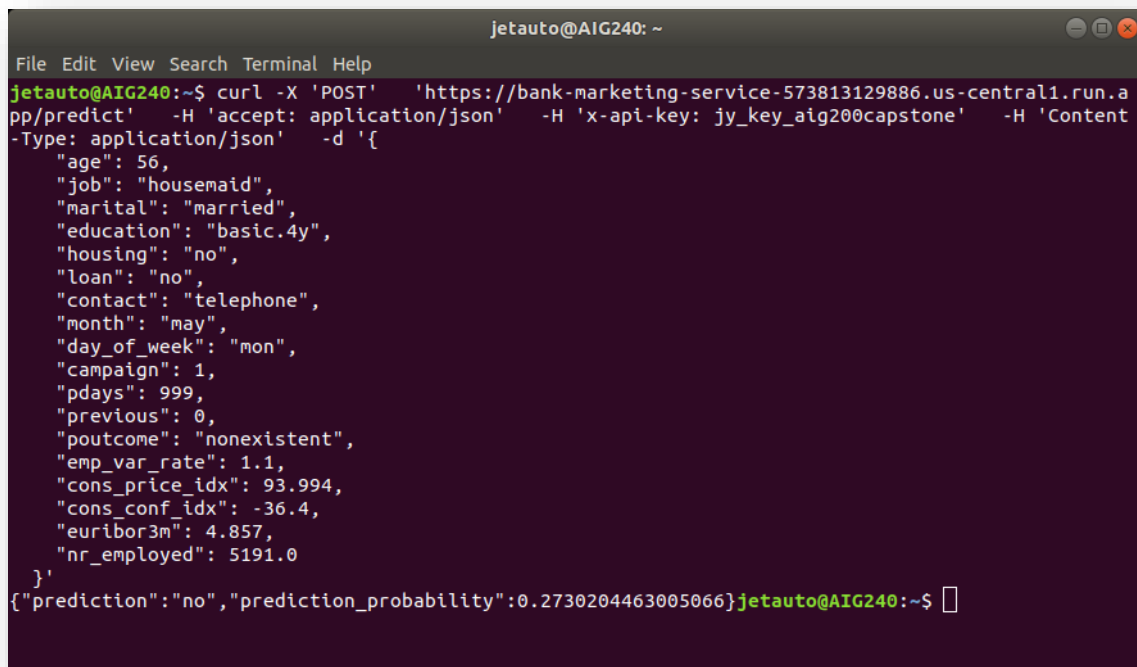
Server response

Code	Details
200	<p>Response body</p> <pre>{   "prediction": "no",   "prediction_probability": 0.2730204463005066 }</pre> <p>Response headers</p> <pre>alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000 content-length: 63 content-type: application/json date: Sun, 22 Jun 2025 20:50:49 GMT server: Google Frontend x-cloud-trace-context: dd2807389f6137020e4cdfef7614b946;o=1</pre>

Responses

## FastAPI Interactive Documentation Response

The following screenshot shows a successful test of the live endpoint using CURL.

A terminal window titled 'jetauto@AIG240: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows a curl command being executed. The command is: curl -X 'POST' 'https://bank-marketing-service-573813129886.us-central1.run.app/predict' -H 'accept: application/json' -H 'x-api-key: jy\_key\_aig200capstone' -H 'Content-Type: application/json' -d '{ "age": 56, "job": "housemaid", "marital": "married", "education": "basic.4y", "housing": "no", "loan": "no", "contact": "telephone", "month": "may", "day\_of\_week": "mon", "campaign": 1, "pdays": 999, "previous": 0, "poutcome": "nonexistent", "emp\_var\_rate": 1.1, "cons\_price\_idx": 93.994, "cons\_conf\_idx": -36.4, "euribor3m": 4.857, "nr\_employed": 5191.0 }'. The output of the command is: {"prediction": "no", "prediction\_probability": 0.2730204463005066}. The prompt 'jetauto@AIG240:~\$' is visible at the end of the line.

```
jetauto@AIG240: ~  
File Edit View Search Terminal Help  
jetauto@AIG240:~$ curl -X 'POST' 'https://bank-marketing-service-573813129886.us-central1.run.a  
pp/predict' -H 'accept: application/json' -H 'x-api-key: jy_key_aig200capstone' -H 'Content  
-Type: application/json' -d '{  
  "age": 56,  
  "job": "housemaid",  
  "marital": "married",  
  "education": "basic.4y",  
  "housing": "no",  
  "loan": "no",  
  "contact": "telephone",  
  "month": "may",  
  "day_of_week": "mon",  
  "campaign": 1,  
  "pdays": 999,  
  "previous": 0,  
  "poutcome": "nonexistent",  
  "emp_var_rate": 1.1,  
  "cons_price_idx": 93.994,  
  "cons_conf_idx": -36.4,  
  "euribor3m": 4.857,  
  "nr_employed": 5191.0  
}'  
{"prediction": "no", "prediction_probability": 0.2730204463005066} jetauto@AIG240:~$
```

Access with CURL

## 8 Conclusion and Future Improvements

This project successfully completed an end-to-end process of deploying a machine learning model, from data analysis to a live, scalable cloud-based API. The final deployed service effectively predicts customer subscription likelihood and is configured to prioritize recall, aligning with business goals for marketing outreach.

For future improvements, the following could be considered:

- **Hyperparameter Optimization:** Use automated tools like *KerasTuner* or *Optuna* to systematically search for an even better model architecture.
- **CI/CD Pipeline:** Implement a Continuous Integration/Continuous Deployment pipeline to automate the testing, building, and deployment of new model versions.
- **Advanced Monitoring:** Integrate more advanced logging and monitoring to track API performance, latency, and prediction drift over time.
- **A/B Testing:** Deploy multiple model versions simultaneously to A/B test their real-world impact on campaign conversion rates.

## 9 References

1. Mazlum Tosun. (2023, May 26). Cloud Run service with a Python module: FastAPI and Uvicorn. Medium. <https://medium.com/google-cloud/cloud-run-service-with-a-python-module-fastapi-and-uvicorn-24c94090a008>
2. Google Cloud. (n.d.). Deploy a Python service to Cloud Run. Google Cloud. <https://cloud.google.com/run/docs/quickstarts/build-and-deploy/deploy-python-service>