Bank Marketing MLOps Demo (Direct-Use API)

End-to-end classification workflow that predicts whether a banking customer will subscribe to a term deposit.
This fork removes the login flow: the predictor UI is available at `/app`, and the `/predict` API can be called directly.

The API **engineers helper features at inference time** (so your JSON only needs the raw fields).

Tech Stack

- Python 3.11+
- Pandas, NumPy, scikit-learn
- MLflow (latest)
- FastAPI + Uvicorn (served with Gunicorn in Azure)
- Azure App Service (Linux, Basic B1)

Project Layout (high level)

```
data/                # raw dataset provided
models/              # trained MLflow model (created by training)
dist/                # packaged model zip (created by packaging)
src/                 # config, data, features, training pipeline
app/                 # FastAPI app (main.py + templates)
scripts/             # CLI: train, package, setup_model, run_api
azure/               # Dockerfile, entrypoint, startup.txt, deploy scripts
requirements.txt
```

Key bits: - `app/main.py` exposes: - `GET /` → marketing landing page - `GET /app` → predictor dashboard (no auth) - `POST /predict` → scoring endpoint (Pydantic schema below) - `src/features/engineering.py` defines engineered columns used by the model.
The API **applies this automatically** before prediction. - `scripts/setup_model.py` downloads/extracts a model when `MODEL_URI` is set (HTTP/HTTPS/Azure Blob/local path/zip).

Local: setup, train, package, serve

```
# 1) env
python -m venv .venv
.\.venv\Scripts\Activate.ps1
```

```
pip install --upgrade pip
pip install -r requirements.txt

# 2) train (logs to ./mlruns; saves model to ./models/bank_marketing_model)
python -m scripts.train_model --experiment-name bank_marketing_classification

# 3) package (creates dist/bank_marketing_model.zip)
python -m scripts.package_model --output-dir dist

# 4) serve locally (expects model in ./models/bank_marketing_model)
uvicorn app.main:app --reload

# UI
#   http://127.0.0.1:8000/app
# API docs
#   http://127.0.0.1:8000/docs
```

JSON request for `/predict`:

```
{
  "age": 42,
  "job": "blue-collar",
  "marital": "married",
  "education": "secondary",
  "default": "no",
  "balance": 1500,
  "housing": "yes",
  "loan": "no",
  "contact": "cellular",
  "day": 15,
  "month": "may",
  "duration": 180,
  "campaign": 2,
  "pdays": 999,
  "previous": 0,
  "poutcome": "unknown"
}
```

The service engineers these features internally: `log_duration`, `log_campaign`, `is_balance_positive`, `has_previous_contact`.

Deployment (CI/CD with GitHub → Azure App Service)

> Replaces the previous “Quick Deploy”/zip-deploy section. This guide sets up **source-based CI/CD**: push to GitHub → build & deploy to your Azure Web App via **GitHub Actions** created by App Service **Deployment Center**.

Prerequisites

- Azure subscription + **Azure CLI** installed and logged in: `az login`
- GitHub account
- A Web App (Linux) and Storage account for hosting the model (create below)
- Local repo checkout of this project

Defaults used by scripts (change as needed):

```
$rg = "rg-bank-marketing"
$st = "stbankmarketing"
$app = "bank-marketing-api"
$container = "models"
```

---

0) One-time Azure resources

From the repo root, run the provisioning script to create: Resource Group, Linux Plan, Web App, Storage account + container.

```
powershell -ExecutionPolicy Bypass -File .\azure\deploy.ps1
```

> If you created resources with different names, update the variables you use in the commands below.

---

1) Package the model and upload to Azure Blob Storage

1. **Package the trained model** (if not already):

   ```
   python -m scripts.package_model --output-dir dist
   ```

   This creates: `dist/bank_marketing_model.zip`.

2. **Get a connection string** and **upload** the model to the `models` container:

```
$CONN = az storage account show-connection-string `
  --name $st --resource-group $rg --query connectionString -o tsv

az storage blob upload `
  --connection-string $CONN `
  --container-name $container `
  --file dist\bank_marketing_model.zip `
  --name bank_marketing_model.zip
```

3. **Create a short-lived SAS** URL for the model:

```
$SAS = az storage blob generate-sas `
  --connection-string $CONN `
  --container-name $container `
  --name bank_marketing_model.zip `
  --permissions r --expiry (Get-Date).AddDays(7).ToString("yyyy-MM-dd") `
  -o tsv

$MODEL_URL = "https://$st.blob.core.windows.net/$container/
bank_marketing_model.zip?$SAS"
```

You can later rotate this SAS and update App Settings without changing the pipeline.

---

2) Configure App Settings & Startup (once)

Set the required environment variables on the Web App and define the startup command:

```
az webapp config appsettings set -g $rg -n $app --settings `
  MODEL_URI="$MODEL_URL" `
  WEBSITES_PORT=8000 `
  SCM_DO_BUILD_DURING_DEPLOYMENT=true `
  MLFLOW_TRACKING_URI="file:/home/site/wwwroot/mlruns"

# Set Startup Command so App Service uses gunicorn via our script
az webapp config set -g $rg -n $app --startup-file "bash azure/startup.txt"
```

Why startup.txt? It ensures dependencies are installed, the model is fetched via `scripts/setup_model.py`, and Gunicorn/Uvicorn is launched consistently on Linux App Service.

---

3) Push local code to a new GitHub repository

1. Initialize and commit (if you haven't already):

```
git init
git add .
git commit -m "Initial commit: bank marketing API"
```

2. Create a GitHub repo (via GitHub UI or CLI) and add it as `origin`:

```
git remote add origin https://github.com/<your-org-or-user>/<your-repo>.git
git branch -M main
git push -u origin main
```

If your local is behind GitHub (fast-forward safely)

Before pushing, bring your branch up to date without merging remote changes into a new merge commit:

```
git pull --rebase --autostash origin main
# autostash saves your uncommitted work, rebases your commits on top of origin/
main,
# then reapplies your local changes. Resolve conflicts if prompted.

git push origin main
```

Notes - Use this when you want a **linear history** (no merge commits from `git pull`). - If you already pushed your branch to GitHub and others pulled it, avoid rebasing unless everyone agrees. - If you prefer a merge-based workflow, you can instead do: `git pull origin main` (no `--rebase`).

---

4) Enable CI/CD from Azure App Service to GitHub (Deployment Center) from Azure App Service to GitHub (Deployment Center)

1. Open **Azure Portal → App Services → *your app* → Deployment Center**.
2. **Source**: choose **GitHub**. Authorize if prompted.
3. Select your **Organization → Repository → Branch** (e.g., `main`).
4. **Build provider**: choose **GitHub Actions**.
5. **Runtime stack**: **Python 3.11**; OS: **Linux**.
6. Click **Save** / **Finish**. Azure will:
7. Create a workflow file in your repo: `.github/workflows/<something>.yml`.
8. Create a GitHub Secret for the publish profile and reference it in the workflow.
9. Trigger the **first CI/CD run** automatically.

If you prefer to add a workflow yourself, use the template below.

---

5) GitHub Actions workflow (auto-generated by Azure App Service Deployment Center)

When you enable CI/CD from **Azure App Service → Deployment Center** and select GitHub as the source with **GitHub Actions** as the build provider, Azure **auto-generates** a workflow file in your repository under `.github/workflows/`. A typical workflow looks like this (generated by Deployment Center):

```
# Docs for the Azure Web Apps Deploy action: https://github.com/Azure/webapps-deploy
# More GitHub Actions for Azure: https://github.com/Azure/actions
# More info on Python, GitHub Actions, and Azure App Service: https://aka.ms/python-webapps-actions

name: Build and deploy Python app to Azure Web App - bank-marketing-api

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read #This is required for actions/checkout

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python version
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'

      # 🛠️ Local Build Section (Optional)

# The following section in your workflow is designed to catch build issues early on the client side, before deployment. This can be helpful for debugging and validation. However, if this step significantly increases deployment time and early detection is not critical for your workflow, you may remove this section to streamline the deployment process.
```

```
      - name: Create and Start virtual environment and Install dependencies
        run: |
          python -m venv antenv
          source antenv/bin/activate
          pip install -r requirements.txt

      # By default, when you enable GitHub CI/CD integration through the Azure
portal, the platform automatically sets the SCM_DO_BUILD_DURING_DEPLOYMENT
application setting to true. This triggers the use of Oryx, a build engine that
handles application compilation and dependency installation (e.g., pip install)
directly on the platform during deployment. Hence, we exclude the antenv virtual
environment directory from the deployment artifact to reduce the payload size.
      - name: Upload artifact for deployment jobs
        uses: actions/upload-artifact@v4
        with:
          name: python-app
          path: |
            .
            !antenv/

      # 🗄️ Opting Out of Oryx Build
      # If you prefer to disable the Oryx build process during deployment,
follow these steps:

# 1. Remove the SCM_DO_BUILD_DURING_DEPLOYMENT app setting from your Azure App
Service Environment variables.
      # 2. Refer to sample workflows for alternative deployment strategies:
https://github.com/Azure/actions-workflow-samples/tree/master/AppService


  deploy:
    runs-on: ubuntu-latest
    needs: build
    permissions:
      id-token: write #This is required for requesting the JWT
      contents: read #This is required for actions/checkout

    steps:
      - name: Download artifact from build job
        uses: actions/download-artifact@v4
        with:
          name: python-app

      - name: Login to Azure
        uses: azure/login@v2
        with:
          client-id: $
{{ secrets.AZUREAPPSERVICE_CLIENTID_599F19C6F97C4EF5B8D9F6CF27266AA9 }}
```

```
          tenant-id: $
{{ secrets.AZUREAPPSERVICE_TENANTID_34AE7BFA6D9C458F88D720290F312701 }}
          subscription-id: $
{{ secrets.AZUREAPPSERVICE_SUBSCRIPTIONID_A2D6D447C4234986A08FA51AE2BD9170 }}

      - name: 'Deploy to Azure Web App'
        uses: azure/webapps-deploy@v3
        id: deploy-to-webapp
        with:
          app-name: 'bank-marketing-api'
          slot-name: 'Production'
```

What Azure sets up for you - The workflow file (name may vary) is committed to your repo by the Deployment Center wizard. - **Authentication**: uses **OpenID Connect (OIDC)** via `azure/login@v2` with secrets added as GitHub Action secrets by the wizard. You don't need to create a publish profile secret for this flow. - **Build**: Oryx runs on the App Service side by default because `SCM_DO_BUILD_DURING_DEPLOYMENT=true` is set in App Settings. The local build step in the workflow is optional and can be removed for faster runs. - **Artifact**: The workflow uploads the repo as an artifact in the `build` job and downloads it in the `deploy` job.

If you later customize this workflow (e.g., add tests, linting, packaging, or multi-env deployments), keep the `azure/login` and `azure/webapps-deploy` steps intact, and ensure `app-name` matches your Web App.

---

6) Verify a deployment

After the workflow completes:

```
az webapp browse -g $rg -n $app
# UI:    https://{app}.azurewebsites.net/app
# Docs:  https://{app}.azurewebsites.net/docs
```

If you change the model location/SAS later, update **MODEL_URI** in App Settings and **Restart** the app:

```
az webapp restart -g $rg -n $app
```

Troubleshooting CI/CD

- **Workflow fails on deploy**
- Ensure the **publish profile secret** is present and referenced by the exact name in the YAML.
- Confirm the `app-name` in the workflow matches your Azure Web App name.

- **502/timeout after deploy**
- Confirm `WEBSITES_PORT=8000` is set and **Startup Command** = `bash azure/startup.txt`.
- First start may take time to install Python deps and download the model.
- **500 "Model not available"**
- SAS expired or `MODEL_URI` incorrect. Rotate SAS and update App Settings, then restart.
- **403 when fetching model**
- SAS read permission or container mismatch. Regenerate SAS with `--permissions r` and correct blob name.
- **500 "columns are missing" / wrong predictions**
- Ensure this fork (which **engineers features inside** `/predict`) is what you deployed.

Configuration

Setting	Purpose	Default / Example
`MODEL_URI`	Remote/Local URI to **zip or directory** with MLflow model. Supports HTTPS & Azure Blob.	`https://stbankmarketing.blob.core.windows.net/models/bank_marketing_model.zip?...`
`MODEL_LOCAL_PATH`	Where the model is placed inside the app container.	`model/bank_marketing_model`
`MLFLOW_TRACKING_URI`	Where MLflow logs (for the app).	`file:/home/site/wwwroot/mlruns`
`WEBSITES_PORT`	Port to listen on in Azure App Service (Linux).	`8000`
`PYTHONPATH`	Optional module path.	`/home/site/wwwroot`

API Summary

- **POST** `/predict` → returns `{{ "probability": float, "prediction": 0|1 }}`
  Request fields: `age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome`.
- **GET** `/app` → browser UI to submit a form and see the prediction.
- **GET** `/docs` → interactive OpenAPI docs.

Notes & Next Steps

- To make `/` go straight to the predictor, change the index route to return `render_dashboard_html()` or add a redirect in `app/main.py`.
- **CI/CD**: You now deploy by **pushing to GitHub**. The workflow will build and publish to Azure automatically.
  Consider a second workflow for the offline ML loop: run `train_model` → evaluate → `package_model` → upload to Blob → update `MODEL_URI` via CLI or Portal.
- Consider storing secrets (SAS, connection strings) in Azure Key Vault and referencing them from App Settings.

Dataset: UCI Bank Marketing – for research/education.
Security: Never commit secrets. Rotate SAS regularly.