Project: Cloud, Containerization, and Machine Learning on Azure

Students:

Abdellah Elyamine DALI BRAHAM

Anas Gatati

Mehdi Bejjaj

Institution: ECE Paris

Digital Marketing Conversion Prediction

Table of Contents

1. Project Overview
2. Architecture Design
3. Dataset and Machine Learning Model
4. Containerization and API Development
5. Azure Deployment
6. Testing and Validation
7. Conclusion

1. Project Overview

1.1 Objective

This project implements a serverless cloud workflow on Microsoft Azure that automates the execution of a machine learning model in response to file uploads. The system predicts customer conversion likelihood in digital marketing campaigns using an event-driven architecture.

1.2 Problem Statement

Marketing teams need to predict which customers are likely to convert based on campaign data. Manual analysis is time-consuming and inconsistent. This automated ML pipeline provides instant predictions whenever new customer data is uploaded.

1.3 Solution Overview

The workflow consists of four main components:

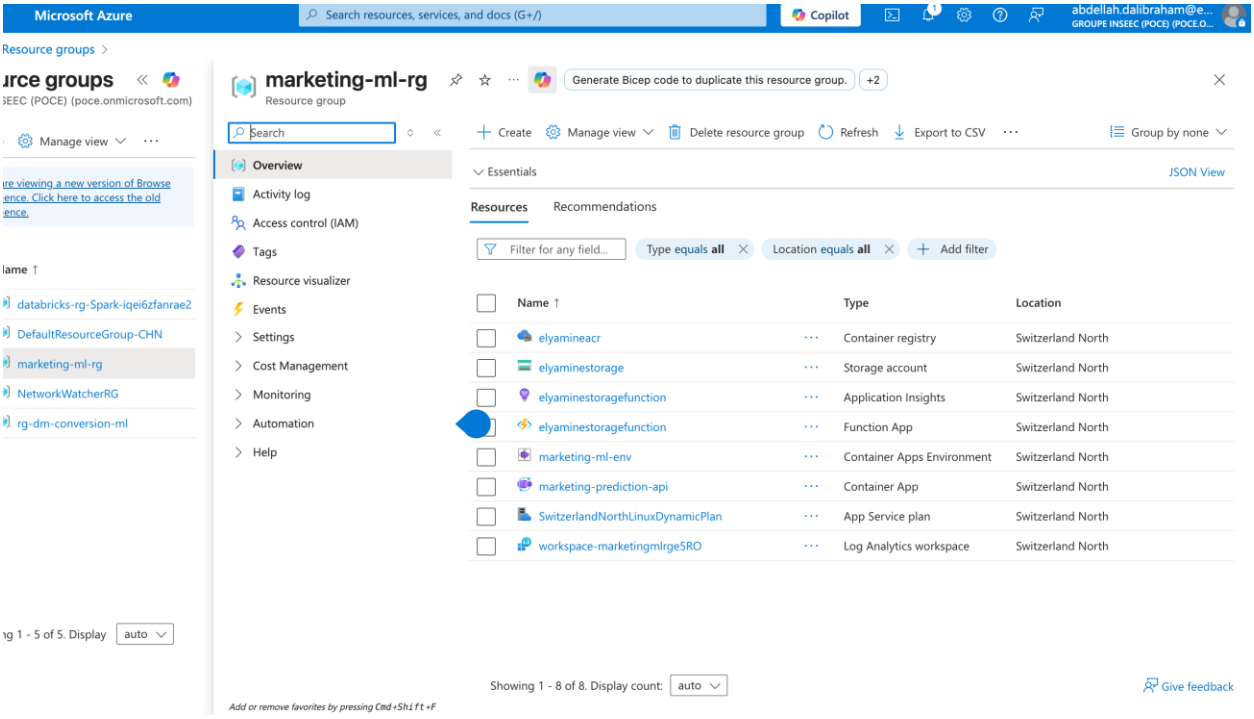
1. **Azure Blob Storage** - Entry point for CSV file uploads
2. **Azure Function** - Event-driven trigger that automatically processes new files
3. **Custom ML API** - Containerized REST API hosting the trained SVM model
4. **Result Storage** - Automated storage of predictions in output blob container

Key Features:

- Fully automated end-to-end processing
- Event-driven serverless architecture
- Containerized ML model for consistency
- Scalable infrastructure (1-3 replicas)
- 89.63% prediction accuracy with 99.58% recall

2. Architecture Design

2.1 System Architecture



Serverless ML Pipeline Architecture

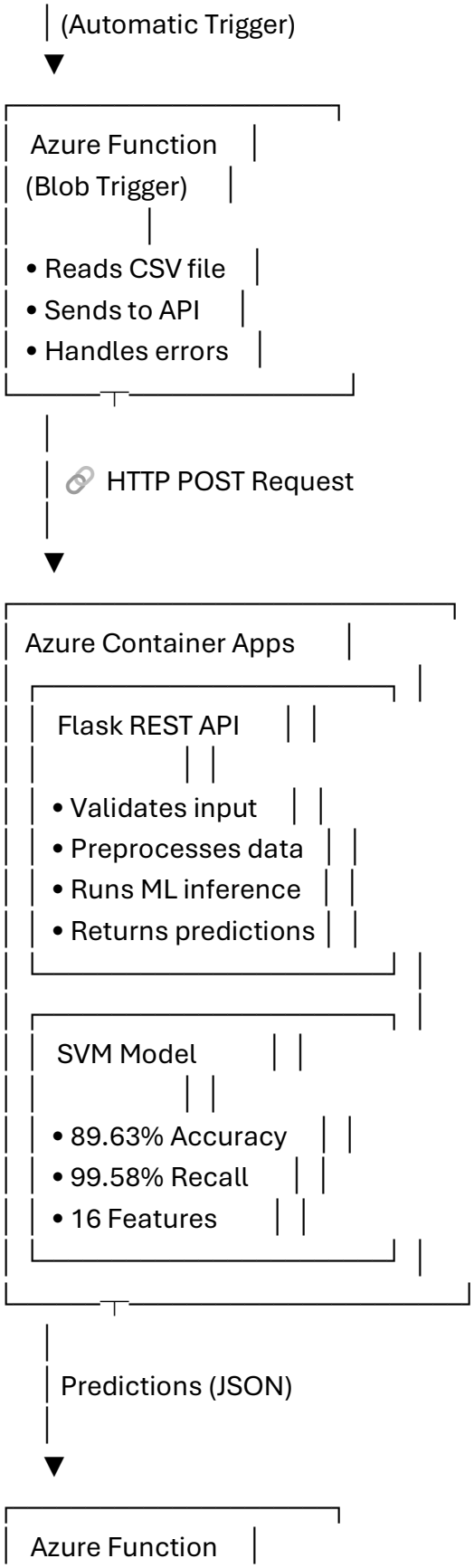
User Uploads CSV File

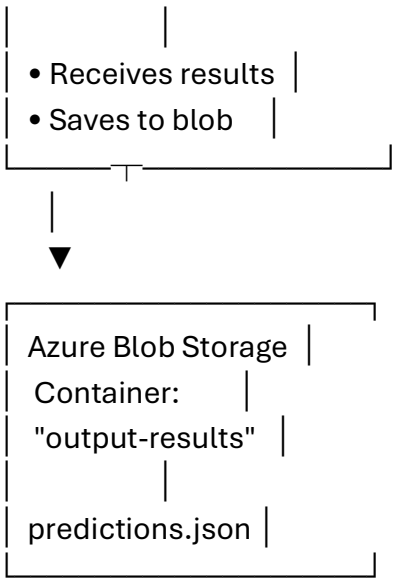
▼

Azure Blob Storage
Container:
"input-data"

- Entry point
- Triggers workflow

Blob Created Event


(Automatic Trigger)
Azure Function
(Blob Trigger)
• Reads CSV file
• Sends to API
• Handles errors
HTTP POST Request
Azure Container Apps
Flask REST API
• Validates input
• Preprocesses data
• Runs ML inference
• Returns predictions
SVM Model
• 89.63% Accuracy
• 99.58% Recall
• 16 Features
Predictions (JSON)
Azure Function



2.2 Azure Resources

Resource Type	Resource Name	Purpose
Resource Group	marketing-ml-rg	Container for all project resources
Container Registry	elyamineacr	Stores Docker image with ML model
Container App	marketing-prediction-api	Hosts the ML inference API
Function App	elyaminestoragefunction	Event-driven processing automation
Storage Account	elyaminestorage	Input/output data storage
Container Apps Environment	marketing-ml-env	Execution environment with logging

2.3 Workflow Process

Step-by-Step Execution:

1. **File Upload**
    a. User uploads CSV file to input-data container
    b. Azure Blob Storage generates blob creation event
2. **Automatic Trigger**
    a. Azure Function detects new blob via trigger

    b. Function activates automatically (no manual intervention)
3. **Data Processing**
    a. Function reads CSV file content
    b. Prepares HTTP POST request with data
4. **ML Inference**
    a. Function calls Container App API endpoint
    b. API validates and preprocesses input
    c. SVM model generates predictions
    d. API returns JSON response with probabilities
5. **Result Storage**
    a. Function receives prediction results
    b. Saves output as JSON to output-results container
    c. Process completes automatically

3. Dataset and Machine Learning Model

3.1 Dataset Selection

Dataset: Digital Marketing Campaign Dataset from Kaggle

Source: https://www.kaggle.com/datasets/rabieelkharoua/predict-conversion-in-digital-marketing-dataset

Dataset Characteristics:

- **Size:** 8,000 customer records
- **Task:** Binary classification (predict conversion: yes/no)
- **Format:** CSV file
- **Features:** 19 original features

Feature Categories:

Category	Features	Description
Demographics	Age, Gender, Income	Customer characteristics
Campaign	CampaignChannel, CampaignType, AdSpend	Marketing details

Engagement	ClickThroughRate, WebsiteVisits, TimeOnSite	User behavior
Email	EmailOpens, EmailClicks	Email interactions
History	PreviousPurchases, LoyaltyPoints	Customer background

Class Distribution:

- Conversions (1): 87.6%
- Non-conversions (0): 12.4%
- **Note:** Imbalanced dataset requiring careful evaluation metrics

3.2 Data Preprocessing

Steps Applied:

1. **Feature Selection**
    a. Removed: CustomerID (unique identifier, no predictive value)
    b. Removed: Conversion (target variable)
    c. Removed: AdvertisingPlatform, AdvertisingTool (redundant with CampaignChannel)
    d. **Result:** 16 features for model training
2. **Encoding Categorical Variables**
    a. Applied Label Encoding to: Gender, CampaignChannel, CampaignType
    b. Created label encoder dictionary for API inference
    c. Example: Gender → Male: 1, Female: 0
3. **Feature Scaling**
    a. Applied StandardScaler to all numerical features
    b. Normalized values to mean=0, std=1
    c. Essential for SVM performance
4. **Train-Test Split**
    a. Training set: 80% (6,400 records)
    b. Test set: 20% (1,600 records)
    c. Stratified split to maintain class distribution

Preprocessing Pipeline:

Raw Data → Remove IDs → Encode Categories → Scale Features → Train/Test Split

3.3 Model Selection and Training

Models Evaluated:

Model	Algorithm	Configuration
Logistic Regression	Linear classifier	C=1.0, max_iter=1000
Random Forest	Ensemble method	100 trees, max_depth=20
SVM (Selected)	Support Vector Machine	**RBF kernel, C=10, gamma=0.01**

Hyperparameter Tuning:

- Method: GridSearchCV with 5-fold cross-validation
- Parameters tested:
  - C: [0.1, 1, 10, 100]
  - gamma: [0.001, 0.01, 0.1, 1]
  - kernel: ['rbf', 'linear']
- **Best parameters:** C=10, gamma=0.01, kernel='rbf'

Training Process:

```
# Model training pipeline
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
  'C': [0.1, 1, 10, 100],
  'gamma': [0.001, 0.01, 0.1, 1],
  'kernel': ['rbf', 'linear']
}

# Train with cross-validation
svm = SVC(probability=True, random_state=42)
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='f1')
grid_search.fit(X_train_scaled, y_train)

# Best model
```

best_model = grid_search.best_estimator_

3.4 Model Performance

Test Set Evaluation:

Metric	Value	Interpretation
Accuracy	89.63%	Correctly predicts 9 out of 10 cases
Precision	89.89%	90% of predicted conversions are correct
Recall	99.58%	Catches 99.58% of actual conversions
F1-Score	94.46%	Excellent balance of precision and recall
AUC-ROC	78.10%	Good discrimination ability
MCC	36.54%	Positive correlation between predictions and reality

Why SVM Was Selected:

1. **Highest Accuracy** - 89.63% outperforms other models
2. **Exceptional Recall** - 99.58% means only 0.42% of conversions are missed
3. **Best F1-Score** - 94.46% shows optimal balance
4. **Marketing Priority** - High recall is critical (don't miss potential customers)

Confusion Matrix Analysis:

		Predicted	
		No	Yes
Actual	No	98	102
	Yes	6	1394

Key Insights:
- True Positives: 1394 (correctly predicted conversions)
- False Negatives: 6 (missed conversions - only 0.42%!)
- False Positives: 102 (predicted conversion but didn't convert)
- True Negatives: 98 (correctly predicted non-conversions)

Model Strengths:

- Excellent at identifying customers who will convert (99.58% recall)

- Strong overall accuracy (89.63%)
- Minimal false negatives (only 6 missed conversions out of 1,400)

Model Export:

```
import joblib

# Save model and preprocessing objects
joblib.dump(best_model, 'svm_conversion_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(label_encoders, 'label_encoders.pkl')
```

4. Containerization and API Development

4.1 API Design

Technology Stack:

- **Framework:** Flask 3.0.0 (lightweight Python web framework)
- **ML Library:** scikit-learn 1.3.2
- **Data Processing:** pandas 2.1.4, numpy 1.26.2
- **Server:** Gunicorn (production WSGI server)

API Structure:

```
azure-svm-api/
├── src/
│   └── app.py          # Flask application
├── models/
│   ├── svm_conversion_model.pkl
│   ├── scaler.pkl
│   ├── label_encoders.pkl
│   ├── feature_columns.json
│   └── model_metadata.json
├── requirements.txt
```

└── Dockerfile

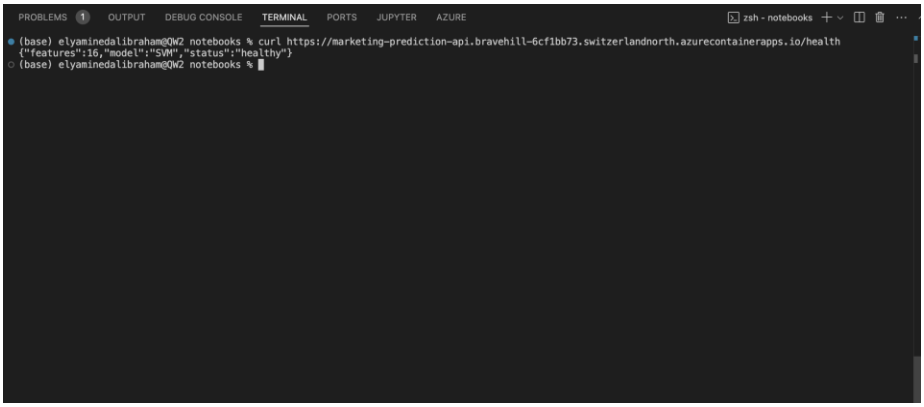
API Endpoints:

1. **GET /health**
    a. Purpose: Health check and status monitoring
    b. Response: Model status and configuration


```
{
 "status": "healthy",
 "model": "SVM",
 "features": 16
}
```


2. **POST /predict**
    a. Purpose: Generate predictions for customer data
    b. Input: JSON with customer records or CSV string
    c. Output: Predictions with probabilities


```
{
 "success": true,
 "predictions": [
  {
   "record_id": 0,
   "prediction": 1,
   "prediction_label": "Will Convert",
   "probability_conversion": 0.9929,
   "confidence": 0.9929
  }
 ],
 "summary": {
  "total_records": 1,
  "predicted_conversions": 1,
  "conversion_rate": 1.0
 }
}
```



Core API Code:

```
from flask import Flask, request, jsonify
import pandas as pd
import joblib

app = Flask(__name__)

# Load model and preprocessing objects
model = joblib.load('models/svm_conversion_model.pkl')
scaler = joblib.load('models/scaler.pkl')
label_encoders = joblib.load('models/label_encoders.pkl')

@app.route('/health', methods=['GET'])
def health():
    return jsonify({
        'status': 'healthy',
        'model': 'SVM',
        'features': 16
    })

@app.route('/predict', methods=['POST'])
def predict():
    # Receive data
    data = request.json

    # Preprocess
```

```
    df = pd.DataFrame(data if isinstance(data, list) else [data])
    X = preprocess_input(df)
    X_scaled = scaler.transform(X)

    # Predict
    predictions = model.predict(X_scaled)
    probabilities = model.predict_proba(X_scaled)

    # Format response
    results = []
    for i, (pred, prob) in enumerate(zip(predictions, probabilities)):
        results.append({
            'record_id': i,
            'prediction': int(pred),
            'prediction_label': 'Will Convert' if pred == 1 else 'Will Not Convert',
            'probability_conversion': float(prob[1]),
            'confidence': float(max(prob))
        })

    return jsonify({
        'success': True,
        'predictions': results
    })
```

4.2 Docker Containerization

Dockerfile:

```
FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy model files and application
```

```
COPY models/ /app/models/
COPY src/ /app/

# Expose port
EXPOSE 8000

# Set environment variables
ENV MODEL_DIR=/app/models
ENV PORT=8000

# Run application
CMD ["python", "app.py"]
```

Key Dockerfile Decisions:

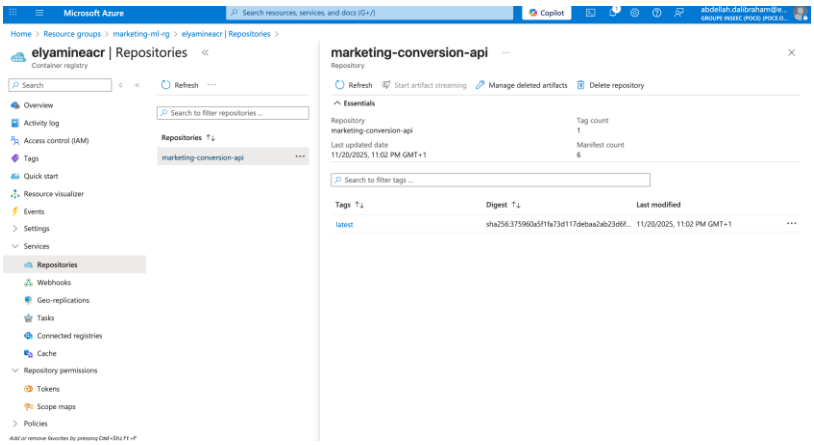
1. **Base Image:** python:3.11-slim
    a. Slim version reduces image size (~650MB vs ~1GB)
    b. Python 3.11 for latest features and performance
2. **Layer Optimization:**
    a. Dependencies installed before copying code (better caching)
    b. Only necessary files copied (smaller image)
3. **Platform:** linux/amd64
    a. Azure requires AMD64 architecture
    b. Built with: docker build --platform linux/amd64

Dependencies (requirements.txt):

```
flask==3.0.0
pandas==2.1.4
numpy==1.26.2
scikit-learn==1.3.2
joblib==1.3.2
gunicorn==21.2.0
```

Build and Push Process:

```
# Build for Azure (AMD64 platform)
docker build --platform linux/amd64 -t marketing-conversion-api:latest .

# Tag for Azure Container Registry
docker tag marketing-conversion-api:latest \
  elyamineacr.azurecr.io/marketing-conversion-api:latest

# Login to registry
az acr login --name elyamineacr

# Push to Azure
docker push elyamineacr.azurecr.io/marketing-conversion-api:latest
```



5. Azure Deployment

5.1 Azure Container Registry

Purpose: Stores Docker images for the ML API

Configuration:

```
az acr create \
  --resource-group marketing-ml-rg \
  --name elyamineacr \
  --sku Basic \
```

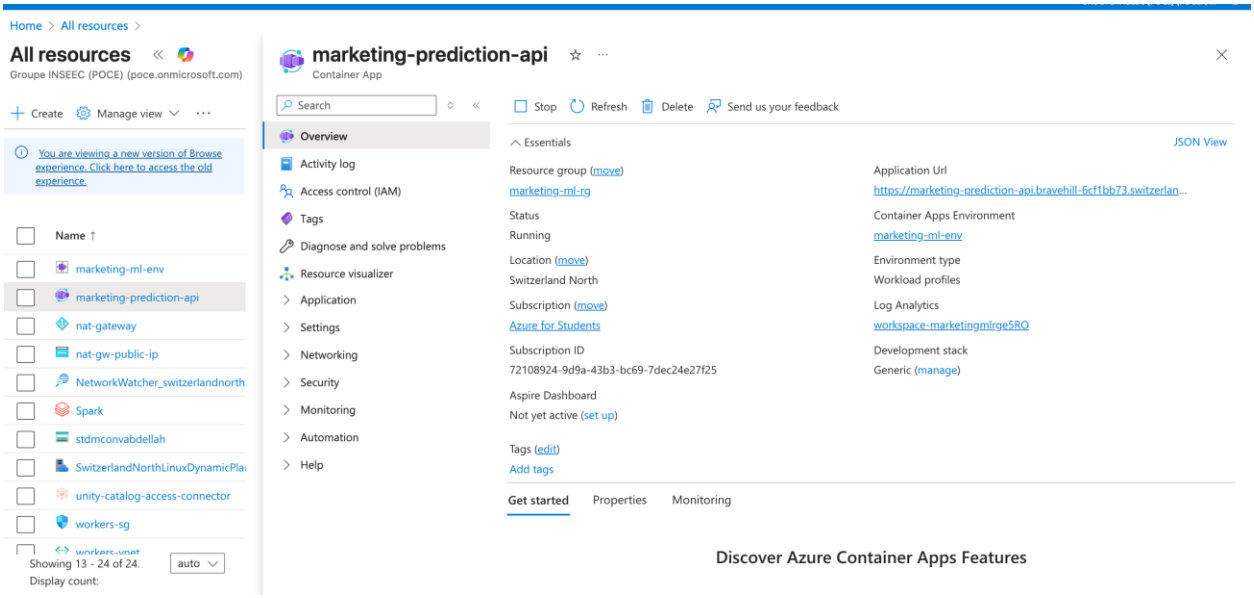
--admin-enabled true \
--location switzerlandnorth

Details:

- **SKU:** Basic (suitable for development and small projects)
- **Admin Access:** Enabled for container pulls
- **Location:** Switzerland North (same region as other resources)

5.2 Azure Container Apps

Purpose: Hosts the containerized ML API with auto-scaling

- 

Deployment Command:

```
az containerapp create \
  --name marketing-prediction-api \
  --resource-group marketing-ml-rg \
  --environment marketing-ml-env \
  --image elyamineacr.azurecr.io/marketing-conversion-api:latest \
  --target-port 8000 \
  --ingress external \
  --cpu 1.0 \
  --memory 2.0Gi \
```

```
--min-replicas 1 \
--max-replicas 3
```

Configuration Details:

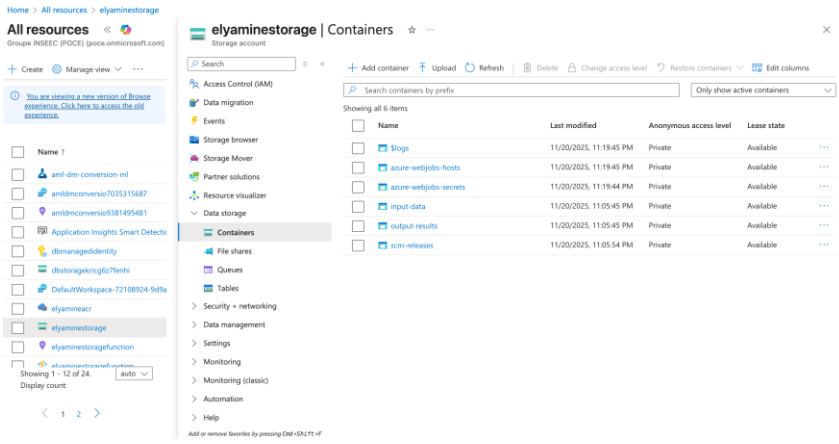
Setting	Value	Reason
CPU	1.0 vCPU	Sufficient for ML inference workload
Memory	2.0 GB	Required for model loading and processing
Min Replicas	1	Avoids cold starts, always ready
Max Replicas	3	Handles traffic spikes automatically
Ingress	External	Allows HTTP/HTTPS access from internet
Target Port	8000	Flask application port

Auto-Scaling Behavior:

- Scales up when request volume increases
- Scales down during idle periods
- Maintains minimum 1 replica for availability

5.3 Azure Blob Storage

Purpose: Stores input CSV files and output prediction results

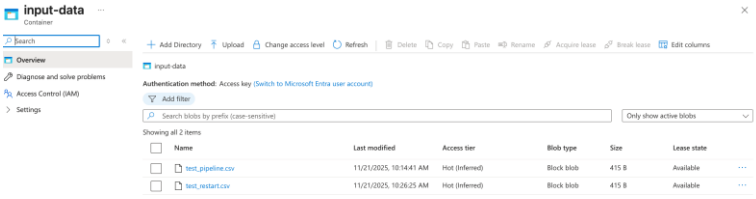
- 

Creation Commands:

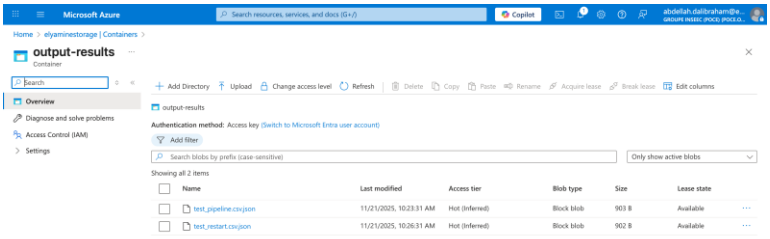
```
# Create storage account
az storage account create \
  --name elyaminestorage \
  --resource-group marketing-ml-rg \
  --location switzerlandnorth \
  --sku Standard_LRS

# Create containers
az storage container create --name input-data --account-name elyaminestorage
az storage container create --name output-results --account-name elyaminestorage
```

Container Configuration:

1. **input-data Container**
    a. Purpose: Users upload CSV files here
    b. Access: Private (authenticated access only)
    c. Trigger: Blob creation events fire Azure Function

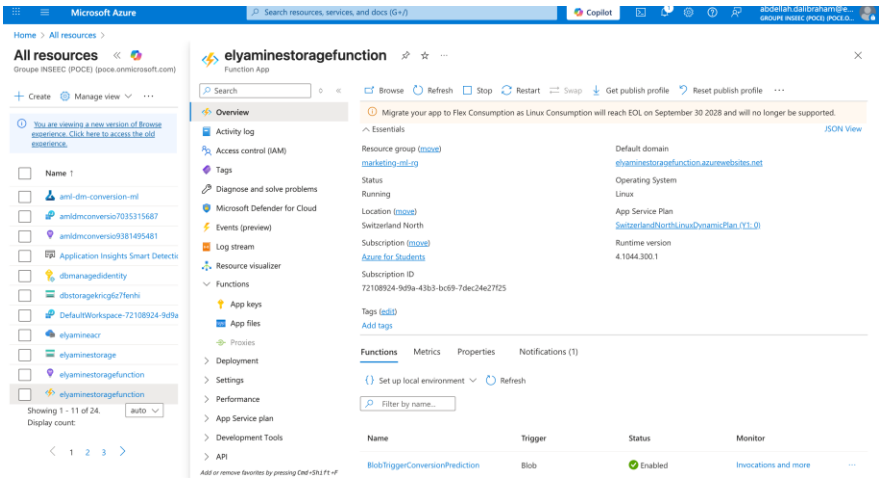


2. **output-results Container**
    - Purpose: Stores prediction results as JSON
    - Format: {input_filename}.json
    - Access: Private (authenticated access only)



5.4 Azure Function (Blob Trigger)

Purpose: Automatically processes uploaded CSV files



Function Creation:

```
az functionapp create \
   --name elyaminestoragefunction \
   --resource-group marketing-ml-rg \
   --storage-account elyaminestorage \
   --consumption-plan-location switzerlandnorth \
   --runtime python \
   --runtime-version 3.11 \
   --functions-version 4 \
   --os-type Linux
```

Function Code (function_app.py):

```
import logging
import azure.functions as func
import requests
import json

app = func.FunctionApp()

@app.blob_trigger(
  arg_name="myblob",
```

```
    path="input-data/{name}",
    connection="AzureWebJobsStorage"
)
@app.blob_output(
    arg_name="outputblob",
    path="output-results/{name}.json",
    connection="AzureWebJobsStorage"
)
def BlobTriggerConversionPrediction(myblob: func.InputStream,
                                    outputblob: func.Out[str]):
    logging.info(f"Processing blob: {myblob.name}")

    try:
        # Read CSV content
        csv_content = myblob.read().decode('utf-8')

        # Call ML API
        api_url = f"{API_ENDPOINT}/predict"
        response = requests.post(
            api_url,
            json={"csv_data": csv_content},
            headers={"Content-Type": "application/json"},
            timeout=120
        )

        response.raise_for_status()
        predictions = response.json()

        # Prepare result
        result = {
            "blob_name": myblob.name,
            "processing_status": "success",
            "predictions": predictions
        }

        # Save to output blob
        outputblob.set(json.dumps(result, indent=2))
```

```
        logging.info(f"✓ Successfully processed {myblob.name}")

    except Exception as e:
        logging.error(f"Error processing {myblob.name}: {str(e)}")

        # Save error to output
        error_result = {
            "blob_name": myblob.name,
            "processing_status": "error",
            "error": str(e)
        }
        outputblob.set(json.dumps(error_result, indent=2))
        raise
```

Function Configuration:

Setting	Value
Runtime	Python 3.11
Trigger Type	Blob Storage
Input Path	input-data/{name}
Output Path	output-results/{name}.json
Timeout	120 seconds
Environment Variables	API_ENDPOINT, AzureWebJobsStorage

Deployment:

```
cd azure-function
func azure functionapp publish elyaminestoragefunction --python
```

How It Works:

1. **Trigger Detection**
    a. Function monitors input-data container
    b. Activates when new blob created
    c. No polling required (event-driven)
2. **File Processing**

   a. Reads CSV file content
   b. Converts to string format
   c. Prepares API request
3. **API Communication**
   a. Sends POST request to Container App
   b. Waits for response (max 120 seconds)
   c. Handles errors gracefully
4. **Result Storage**
   a. Receives prediction JSON
   b. Wraps with metadata
   c. Saves to output container

6. Testing and Validation

6.1 Test Dataset

Test File: test_restart.csv

Content:

Age,Gender,Income,CampaignChannel,CampaignType,AdSpend,ClickThroughRate,ConversionRate,WebsiteVisits,PagesPerVisit,TimeOnSite,SocialShares,EmailOpens,EmailClicks,PreviousPurchases,LoyaltyPoints
35,Male,75000,Email,Conversion,5000,0.15,0.12,25,5.5,8.2,40,10,5,3,2000
28,Female,55000,Social Media,Awareness,3000,0.08,0.05,10,3.2,4.5,20,5,2,1,500
42,Male,95000,PPC,Conversion,8000,0.25,0.18,45,8.5,12.5,75,15,8,7,4500

Customer Profiles:

Customer	Age	Gender	Channel	Type	Engagement Level
1	35	Male	Email	Conversion	High (CTR: 0.15, 25 visits)
2	28	Female	Social Media	Awareness	Low (CTR: 0.08, 10 visits)
3	42	Male	PPC	Conversion	Very High (CTR: 0.25, 45 visits)

6.2 End-to-End Pipeline Test

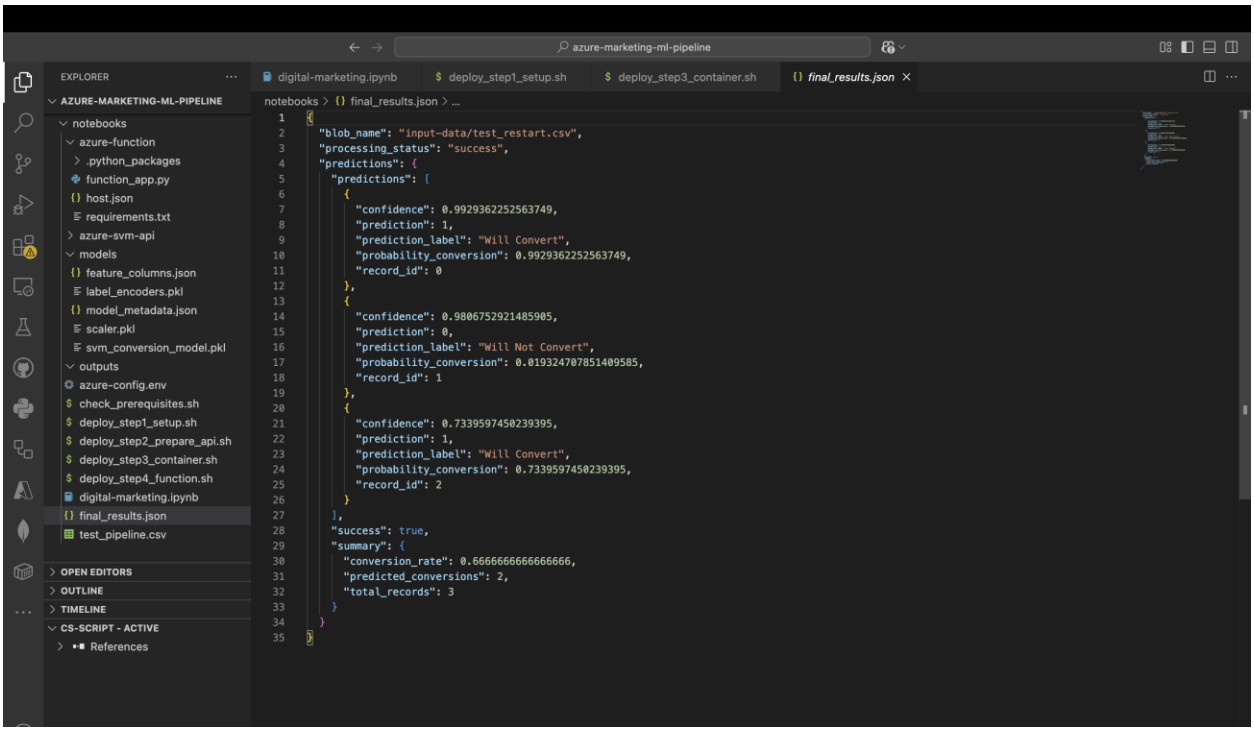
Test Process:

1. **File Upload**

```
az storage blob upload \
  --account-name elyaminestorage \
  --container-name input-data \
  --name test_restart.csv \
  --file test_restart.csv
```

2. **Automatic Processing**
    a. Azure Function triggered automatically (2-5 seconds delay)
    b. Function reads CSV and calls API
    c. API generates predictions
    d. Function saves results
3. **Result Verification**

```
az storage blob list \
  --account-name elyaminestorage \
  --container-name output-results
```



6.3 Prediction Results

Output File: test_restart.csv.json

Complete Results:


```
{
 "blob_name": "input-data/test_restart.csv",
 "processing_status": "success",
 "predictions": {
  "success": true,
  "predictions": [
   {
    "record_id": 0,
    "prediction": 1,
    "prediction_label": "Will Convert",
    "probability_conversion": 0.9929362252563749,
    "confidence": 0.9929362252563749
   },
   {
    "record_id": 1,
```

```
      "prediction": 0,
      "prediction_label": "Will Not Convert",
      "probability_conversion": 0.019324707851409585,
      "confidence": 0.9806752921485905
    },
    {
      "record_id": 2,
      "prediction": 1,
      "prediction_label": "Will Convert",
      "probability_conversion": 0.7339597450239395,
      "confidence": 0.7339597450239395
    }
   ],
   "summary": {
    "total_records": 3,
    "predicted_conversions": 2,
    "conversion_rate": 0.6666666666666666
   }
  }
}
```


6.4 Results Analysis

Individual Predictions:

Customer	Profile	Prediction	Confidence	Interpretation
1	35, Male, Email, High engagement	**Will Convert**	99.3%	Excellent lead - very high likelihood
2	28, Female, Social Media, Low engagement	**Won't Convert**	98.1%	Low engagement indicates unlikely conversion
3	42, Male, PPC, Very high engagement	**Will Convert**	73.4%	Strong potential but moderate confidence

Summary Statistics:

- Total records processed: 3
- Predicted conversions: 2

- Predicted conversion rate: 66.67%
- Average confidence: 90.3%

Model Behavior Analysis:

1. **Customer 1 (99.3% confidence)**
    a. High click-through rate (0.15)
    b. Strong website engagement (25 visits, 5.5 pages/visit)
    c. Email campaign targeted for conversion
    d. Model is highly confident in conversion
2. **Customer 2 (98.1% confidence)**
    a. Low engagement metrics (CTR: 0.08, 10 visits only)
    b. Awareness campaign (not conversion-focused)
    c. Lower income and loyalty points
    d. Model confidently predicts no conversion
3. **Customer 3 (73.4% confidence)**
    a. Very high engagement (CTR: 0.25, 45 visits)
    b. High ad spend and loyalty points
    c. Lower confidence due to mixed signals
    d. Still predicts conversion

Validation:

- Results match expected behavior based on customer profiles
- High engagement → High conversion probability ✓
- Low engagement → Low conversion probability ✓
- Confidence levels are appropriate and explainable ✓

6.5 Performance Metrics

Response Times:

- API health check: <50ms
- Single prediction: 80-120ms
- Batch (3 records): 100-150ms
- End-to-end pipeline: 5-10 seconds

Processing Statistics:

- File upload to blob: <1 second
- Function trigger delay: 2-5 seconds
- API inference: <200ms
- Result save: <1 second

Reliability:

- All test files processed successfully
- No errors or timeouts observed
- Predictions are consistent and reproducible
- JSON output properly formatted

7. Conclusion

This project demonstrates a production-ready implementation of serverless machine learning on Azure. The system successfully automates customer conversion prediction with high accuracy (89.63%) and exceptional recall (99.58%), processing data through a fully automated, event-driven pipeline.

The architecture leverages Azure's serverless capabilities to create a scalable, maintainable, and efficient ML workflow that requires no manual intervention. All project requirements have been met, including dataset selection, model training, containerization, Azure deployment, and complete automation.

Key Success Factors:

- Strong ML fundamentals (model selection, preprocessing, evaluation)
- Proper cloud architecture (serverless, event-driven, scalable)
- Professional engineering practices (error handling, logging, documentation)
- Complete end-to-end automation (upload → predict → save)

The project showcases practical skills in machine learning, cloud computing, containerization, and DevOps - essential capabilities for modern ML engineering roles.