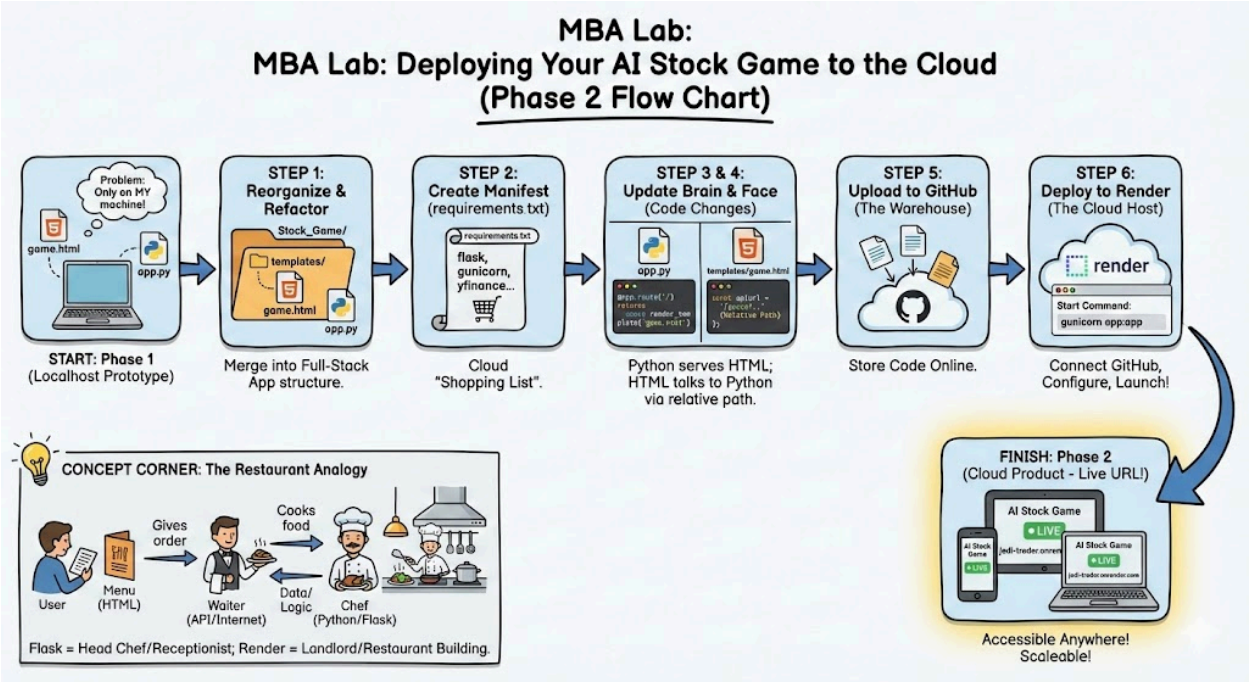
Deploying Your AI Stock Game to the Cloud

Module: Cloud Architecture & Product Deployment

Target Platform: Render (PaaS)

Time Required: 40–50 Minutes

Github: https://github.com/swanlandjack/mba-stock-game



1. Executive Summary & Background

The Business Problem

In the previous lab, you built a working prototype of a "Stock Judgment Engine" on your laptop. However, this prototype has a critical weakness: **it lives and dies with your specific machine.**

- If you close your laptop, the app goes offline.
- Your classmates cannot access it from their phones.

- You cannot scale it to thousands of users.

The Solution: Cloud Deployment

To turn a "Script" into a "Product," we must move it to the Cloud. We will use **Render**, a cloud platform that manages the server infrastructure for us.

What you will do today:

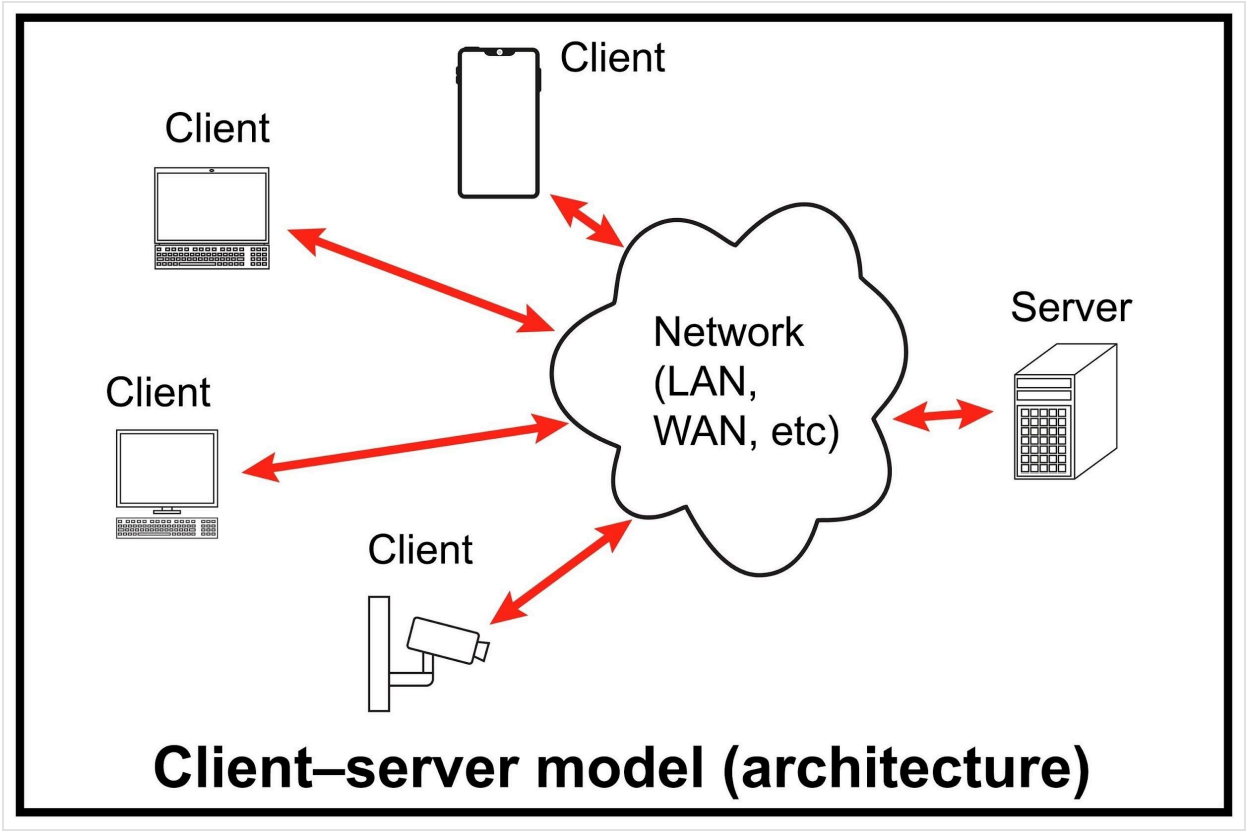
1. **Refactor:** Merge your separate Python and HTML files into a single "Full-Stack Web Application."
2. **Containerize:** Create a "manifest" (`requirements.txt`) so the cloud knows what software your app needs.
3. **Deploy:** Upload your code to **GitHub** and connect it to **Render** to launch a live, public URL.

---

2. Methodology: The Architecture Pivot

We are changing how the application works.

- **Phase 1 (Local):** You opened `game.html` manually. It talked to a separate Python window.
- **Phase 2 (Cloud):** The Python server will act as the **host**.
    - When a user visits your URL, Python serves the HTML page.
    - When the user clicks a button, the HTML page talks back to the *same* Python server to get data.



- Shutterstock

---

3. Step-by-Step Implementation Guide

Step 1: Reorganize Your Folder Structure

Cloud servers are very strict about organization. Flask (our web framework) requires HTML files to be in a specific folder.

1. Open your `Stock_Game` folder.
2. Create a **new sub-folder** named **`templates`**.
3. **Move** your existing `game.html` file **inside** the `templates` folder.

✅ Verify Your Structure:

```
Plaintext

Stock_Game/
├── app.py
├── requirements.txt      (We will create this next)
└── templates/
    └── game.html
```

Step 2: Create the Manifest (`requirements.txt`)

The cloud computer is brand new and empty. We must provide a shopping list of libraries it needs to install.

1. Create a new file in the main `Stock_Game` folder named `requirements.txt`.
2. Paste **exactly** these 4 lines:
3. Plaintext

```
flask
flask-cors
yfinance
gunicorn
```

4.
5. *(**Note:** `gunicorn` is the production server tool. Without it, the app will fail.)*

Step 3: Update the "Brain" (`app.py`)

We need to update the Python code to serve the HTML file. Replace your **entire** `app.py` with this code:

Python

```
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
import yfinance as yf
import os

# Initialize Flask
# We tell it to look for HTML files inside the 'templates' folder
app = Flask(__name__, template_folder='templates')
CORS(app)

# --- ROUTE 1: THE HOMEPAGE (Serving the Interface) ---
@app.route('/')
def home():
    # This sends the game.html file to the user's browser
    return render_template('game.html')

# --- ROUTE 2: THE API (Serving the Logic) ---
@app.route('/guess', methods=['GET'])
def judge_guess():
```

```
    ticker = request.args.get('ticker', 'AAPL').upper()
    prediction = request.args.get('prediction', 'UP').upper()

    try:
        # 1. Fetch Real Data
        stock = yf.Ticker(ticker)
        hist = stock.history(period="5d")
        if hist.empty: return jsonify({"error": "Stock not found"}), 404

        # 2. Calculate Logic
        current = hist['Close'].iloc[-1]
        prev = hist['Close'].iloc[-2]
        change_pct = ((current - prev) / prev) * 100
        direction = "UP" if change_pct > 0 else "DOWN"

        # 3. Determine Result
        result = "WIN" if prediction == direction else "LOSS"

        return jsonify({
            "result": result,
            "details": f"{ticker} moved {direction} by {round(change_pct, 2)}%"
        })
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    # Cloud servers assign a random port. We must read it from the 'PORT' environment variable.
    port = int(os.environ.get("PORT", 5000))
    app.run(host='0.0.0.0', port=port)
```

Step 4: Update the "Face" (templates/game.html)

We need to change how the JavaScript finds the API. Instead of looking for localhost, it should look for "the same server I am currently on."

1. Open templates/game.html.
2. Find the line defining const apiUrl = ... (approx. line 118).
3. **Replace it** with this relative path version:

```
JavaScript

// OLD LINE: const apiUrl = `http://127.0.0.1:5001/guess?...`;

// NEW LINE (Relative Path):
const apiUrl = `/guess?ticker=${ticker}&prediction=${prediction}`;
```

Step 5: Upload to GitHub (The Warehouse)

Render cannot pull files from your laptop. It pulls from GitHub.

1. Log in to **GitHub.com**.
2. Click **(+) New Repository** (Top right).
3. Name it `mba-stock-game` and click **Create**.
4. Click the link **"uploading an existing file"**.
5. **Drag and drop** your `app.py`, `requirements.txt`, and the `templates` folder.
6. Click **Commit changes**.

Step 6: Deploy to Render (The Cloud Host)

1. Go to **dashboard.render.com** and log in with GitHub.
2. Click **New +** > **Web Service**.
3. Select **"Build and deploy from a Git repository"** and click **Next**.
4. Find your `mba-stock-game` repo and click **Connect**.
5. **Configure the Settings:**
    - **Name:** `jedi-trader-[yourname]` (This will be your URL).
    - **Runtime: Python 3**.
    - **Build Command:** `pip install -r requirements.txt` (Default).
    - **Start Command:** `gunicorn app:app`
        - *Critically Important: This tells the production server how to start.*
    - **Instance Type:** Free.
6. Click **Create Web Service**.

Wait 2-3 minutes. When the logs say "Your service is live," click the URL at the top of the page. You are live!

---

4. Concepts Explained (The "Why")

What is Flask? (The Receptionist)

Flask is the software that answers the "phone" when someone types your URL.

- **Without Flask:** Python is just a calculator sitting on a hard drive.
- **With Flask:** Python becomes a **Web Server** that can accept requests, route them to the right function, and return answers to the browser.

What is Render? (The Landlord)

Render provides the **infrastructure**.

- It gives you a computer that is always on.
- It handles security (HTTPS).
- It gives you a public address (URL).
- It installs Python for you based on your `requirements.txt`.

How do they interact? (The Restaurant Analogy)

1. **The Menu (HTML):** The customer (User) looks at the screen and picks "Price Up" (Button).
2. **The Waiter (Internet/API):** Takes that request (`/guess`) to the kitchen.
3. **The Chef (Python/Flask):** receives the order, cooks the data (calculates stock moves), and puts the result on a plate (JSON).
4. **Service:** The waiter brings the result back to the table, and the Menu updates to show "You Win!"

---

5. FAQ & Troubleshooting

Q: Render says "Build Failed" immediately.

- **Check:** Did you create the `requirements.txt` file? Is it spelled correctly? Does it contain `gunicorn`?

Q: The site says "Internal Server Error" when I open it.

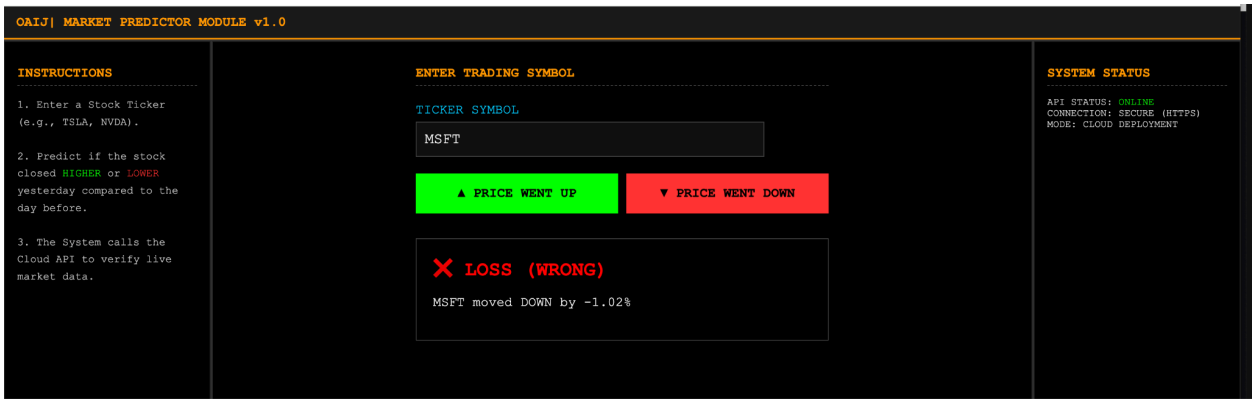
- **Check:** Click the **Logs** tab in Render.
    - If you see `ModuleNotFoundError: No module named 'flask'`, your requirements file is empty or missing.
    - If you see `TemplateNotFound: game.html`, you didn't put the HTML file inside the `templates` folder.

Q: The game loads, but the buttons don't work.

- **Check:** Open your browser console (Right Click > Inspect > Console).
    - If you see a 404 error for `/guess`, your `app.py` might be missing the `@app.route('/guess')` section.
    - If you see a "Mixed Content" error, ensure your API URL is relative (`/guess...`) and not hardcoded to `http://localhost`.

Q: Can I edit the code after deploying?

- **Yes.** Edit the files on your laptop, drag them to GitHub again to "Commit changes," and Render will automatically detect the update and re-deploy your site.


OAIJ| MARKET PREDICTOR MODULE v1.0
INSTRUCTIONS
1. Enter a Stock Ticker (e.g., TSLA, NVDA).
2. Predict if the stock closed HIGHER or LOWER yesterday compared to the day before.
3. The System calls the Cloud API to verify live market data.
ENTER TRADING SYMBOL
TICKER SYMBOL
MSFT
▲ PRICE WENT UP
▼ PRICE WENT DOWN
✕ LOSS (WRONG)
MSFT moved DOWN by -1.02%
SYSTEM STATUS
API STATUS: ONLINE
CONNECTION: SECURE (HTTPS)
MODE: CLOUD DEPLOYMENT