Macro Risk AI Agent - Deployment Guide

1. Project Overview

This document details the deployment procedures for the Macro Risk AI Agent, a containerised SQL-based Large Language Model (LLM) application. The system is designed to analyse UK Insolvency data, validate findings against ground-truth logic, and automatically archive audit reports to a simulated AWS S3 environment.

1.1 Architecture Design

The solution follows a strict Decoupled Cloud-Native pattern, separating the execution logic from the persistence layer.

- Compute Layer:
  - **Component:** macro-risk-agent Docker Container.
  - **Function:** Runs the Python runtime, holds the analyst.py logic, manages the SQLite database, and executes the LangChain agent.
  - **Equivalent in Prod:** AWS ECS (Elastic Container Service) or AWS Lambda.
- Storage Layer:
  - **Component:** localstack Docker Container.
  - **Function:** Simulates the AWS S3 API. It acts purely as an object store for final artifacts (Audit Reports). It does not execute code.
  - **Equivalent in Prod:** AWS S3 (Simple Storage Service).

1.2 Cloud Technology Choice: LocalStack

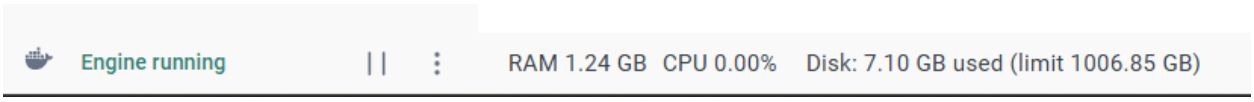
A critical component of this architecture is localstack, a fully functional cloud emulator.

- What it is?
  - LocalStack runs inside a Docker container and mocks the functionality of real AWS Services (such as S3 and Lambda) locally on the host machine. It accepts standard AWS API calls but processes them entirely offline.
- Why it is used?
  - **Cost Efficiency:** Eliminates cloud spend during the Research & Development (R&D) phase. There is no cost for PutObject or GetObject requests.
  - **Data Security:** For sensitive financial or macro-economic modelling, LocalStack ensures that no data leaves the local network. This simulates a high-security, air-gapped environment suitable for defence or banking sectors.
  - **Speed:** Removes latency associated with remote cloud connections, enabling rapid iteration loops for data pipelines.

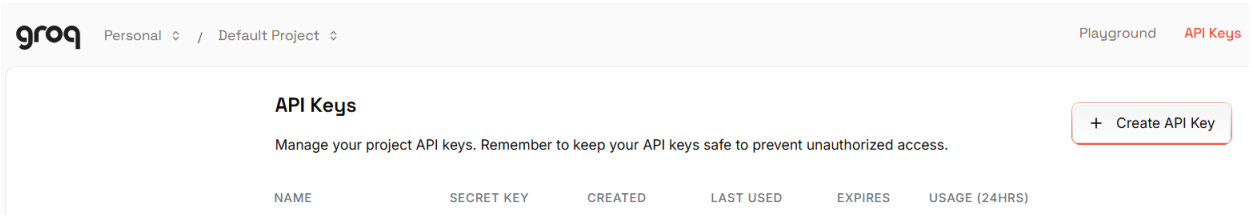
2. Prerequisites

Before initiating the deployment, please ensure the host machine meets the following requirements:

- **Docker Desktop:** Installed and running (green status). This is the engine required to run both the application container and the LocalStack infrastructure.



- **LocalStack CLI:** Installed via pip install localstack. This command-line tool manages the lifecycle of the local cloud emulator.
- **Python 3.9+:** Installed for local script management.
- **Groq API Key:** A valid API key for LLM inference (required for the LangChain agent).



3. Repository Organisation (GitHub)

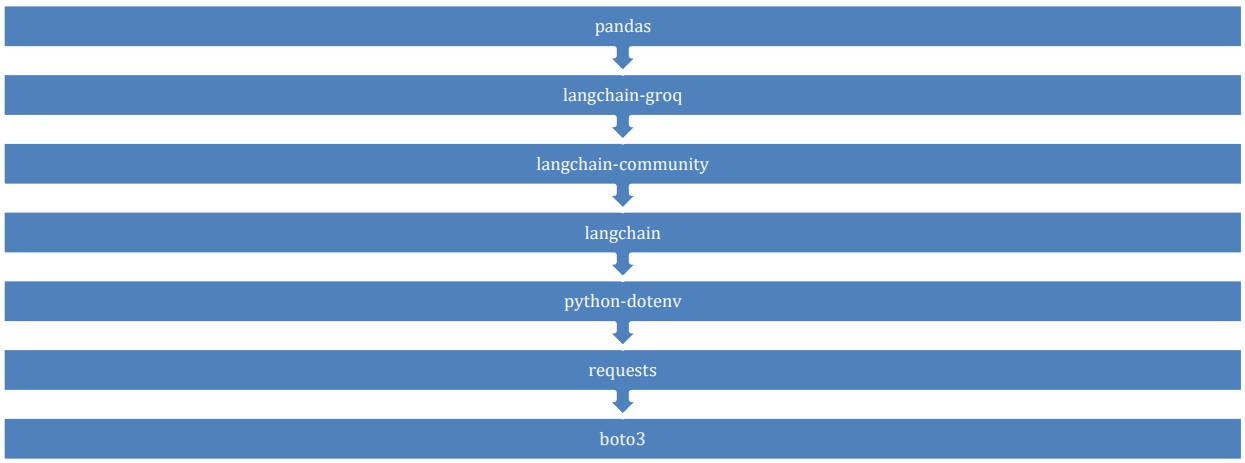
Ensure files are organised as follows before cloning:

```
Macro-Risk-Agent/
├── code/
│   └── docker/
│       └── analyst.py        # Main Application Logic (Compute)
├── data/
│   └── macro_risk.db         # SQLite Database (Insolvency Data)
├── .gitignore                # Excludes .env and __pycache__
├── Dockerfile                # Container Specification
├── README.md                 # General Project Info
└── requirements.txt          # Python Dependencies
```

3.1 Critical Files Content

requirements.txt:

Must include boto3 (AWS SDK for Python). This library allows the Python script to communicate with both real AWS and the LocalStack emulator seamlessly.



Dockerfile:

This defines the container image. It handles dependency installation and file structure setup to match the production environment.

```
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

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

# Copy your project folders into the container
COPY code/ /app/code/
COPY data/ /app/data/
COPY docs/ /app/docs/

# Set Environment Variables
# Ensures Python knows where to look for your modules
ENV PYTHONPATH="/app/code"
# Ensures logs print immediately (crucial for debugging in AWS/Docker)
ENV PYTHONUNBUFFERED=1

# The Start Command
CMD ["python", "code/docker/analyst.py"]
```

4. Local Environment Configuration

4.1 Cloning the Repository

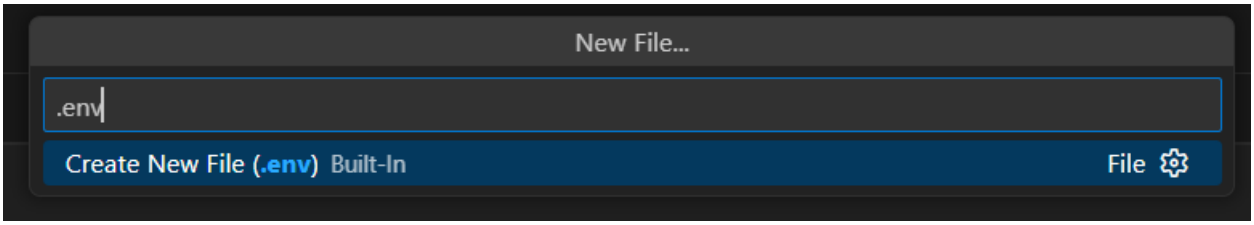
Download the codebase to the local development environment.

- Option A: Git Clone: git clone https://github.com/YOUR_USERNAME/Macro-Risk-Agent.git
  cd Macro-Risk-Agent
- Option B: ZIP Download: Extract ZIP and navigate into the folder via PowerShell
  cd "C:\Users\YourName\Documents\Python\Macro-Risk-Agent"

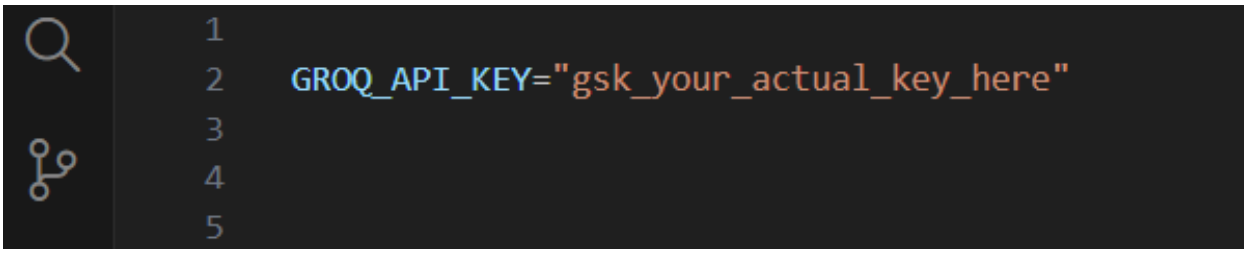
4.2 Security configuration for API keys

Never commit API keys to GitHub. Instead, use a .env file which is excluded from version control.

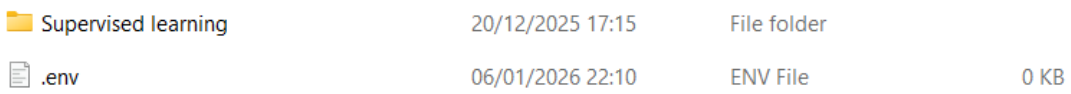
1. Create a file named .env in the project root. If using VScode: newfile > .env



2. Add your API key:
   GROQ_API_KEY=gsk_your_actual_key_here



3. Confirm file is create in windows/mac directory



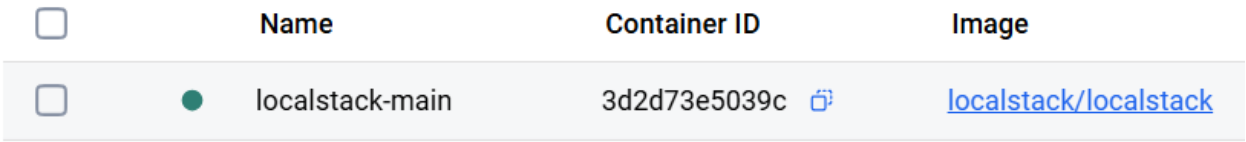
5. Infrastructure Initialisation

5.1 Starting the Local Cloud

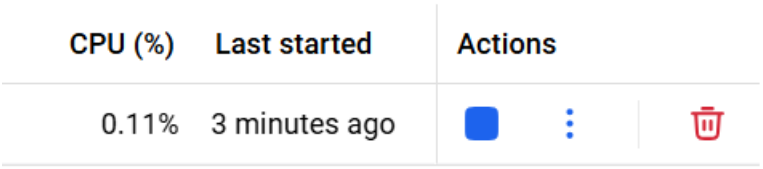
Initiate the LocalStack instance. This spins up a background container that listens on port 4566 (the default LocalStack gateway).

```
> localstack start -d
```

Flag: -d runs the service in detached mode (background), freeing up your terminal for subsequent commands.



Confirmation localstack instance is initialised via docker interface



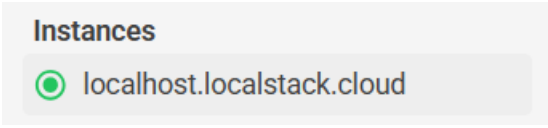
Resources statistics

5.2 Verifying Service Health

Before running the agent, verify that the emulated cloud services are responsive.

localstack status services

- **Success Criteria:** The status for s3 should read available or running.
- **Endpoint Check:** Visit http://localhost:4566/health in a browser. This returns a JSON status report of the local cloud.



Confirmation of initialisation via LocalStack

6. Container Build Process

6.1 Building the Image

Package the application code, database, and dependencies into an immutable Docker image. This ensures the code runs exactly the same on your laptop as it would on a production server.

```
docker build -t macro-risk-agent .
```

- **Tag:** -t macro-risk-agent labels the image for easy reference.
- **Context:** . tells Docker to look in the current directory for the Dockerfile and build context.

Confirmation of build via Windows PowerShell

7. Deployment & Execution

7.1 Running the Agent

Execute the container. This step involves complex networking configuration to bridge the gap between the isolated container and the LocalStack service running on the host.

```
docker run --add-host host.docker.internal:host-gateway --env-file .env macro-risk-agent python code/docker/analyst.py
```

Command Breakdown and Networking Explanation

- --add-host host.docker.internal:host-gateway: Critical Networking Configuration. By default, a Docker container is isolated. This flag modifies the container's internal DNS (hosts file) to map the domain host.docker.internal to the host machine's gateway IP. This allows the Python script *inside* the container to send data *out* to the LocalStack S3 service listening on the host (port 4566).
- --env-file .env: Securely injects the API key at runtime. The key exists in the container's memory only while running, adhering to security principles.
- python code/docker/analyst.py: Overrides the default command to execute the specific analyst script.

8. Verification & Audit

8.1 Terminal Output

Monitor the live logs in the PowerShell window to confirm the pipeline execution.

1. **Agent Analysis:** Observe the LLM answering validation questions via the terminal output.

```
Running Validation Question 1...
Agent Answer: The yearly total Trader Bankruptcies for 2020, 2021, 2022, and 2023 are 2234, 1402, 1227, and 1493 respectively.

Running Validation Question 2...
Agent Answer: 2015: Corp 2853 vs Ind 4404 -> Ind is higher
2016: Corp 2885 vs Ind 3901 -> Ind is higher
2017: Corp 2747 vs Ind 3305 -> Ind is higher
2018: Corp 3090 vs Ind 3661 -> Ind is higher
2019: Corp 2941 vs Ind 3128 -> Ind is higher
2020: Corp 1354 vs Ind 1347 -> Corp is higher
2021: Corp 492 vs Ind 1113 -> Ind is higher
2022: Corp 1968 vs Ind 1143 -> Corp is higher
2023: Corp 2838 vs Ind 1697 -> Corp is higher

Running Validation Question 3...
Agent Answer: The ratio of Total Individual Insolvencies to Total Corporate Insolvencies for the year 2023 is 103434:25162, which simplifies to approximatel
y 4.12:1. This means that for every 1 corporate insolvency, there are approximately 4.12 individual insolvencies.
```

2. **Ground Truth:** Verify that the hard-coded SQL check runs and produces comparative figures.

```
--- GROUND TRUTH (ACTUAL FIGURES) ---
Q1 (Trader Bankruptcies): [(2020, 2234), (2021, 1402), (2022, 1227), (2023, 1493)]

Q2 (Corp Liq vs Ind Forced): [(2015, 2853, 4404), (2016, 2885, 3901), (2017, 2747, 3305), (2018, 3090, 3661), (2019, 2941, 3128), (2020, 1354, 1347), (2021,
 492, 1113), (2022, 1968, 1143), (2023, 2838, 1697)]

Q3 (2023 Ratio): [(4.110722518082824,)]
```

3. **Cloud Upload:** Confirm the final success message: SUCCESS: Report uploaded to S3.

```
--- UPLOADING TO LOCAL CLOUD (S3) ---
/usr/local/lib/python3.9/site-packages/boto3/compat.py:89: PythonDeprecationWarning: Boto3 will no longer support Python 3.9 starting April 29, 2026. To con
tinue receiving service updates, bug fixes, and security updates please upgrade to Python 3.10 or later. More information can be found here: https://aws.ama
zon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
SUCCESS: Report uploaded to S3.
```

8.2 Cloud Artifact Verification

Validate that the report was physically stored in the object storage. This confirms the successful integration between the compute layer (Docker) and Storage layer (LocalStack).

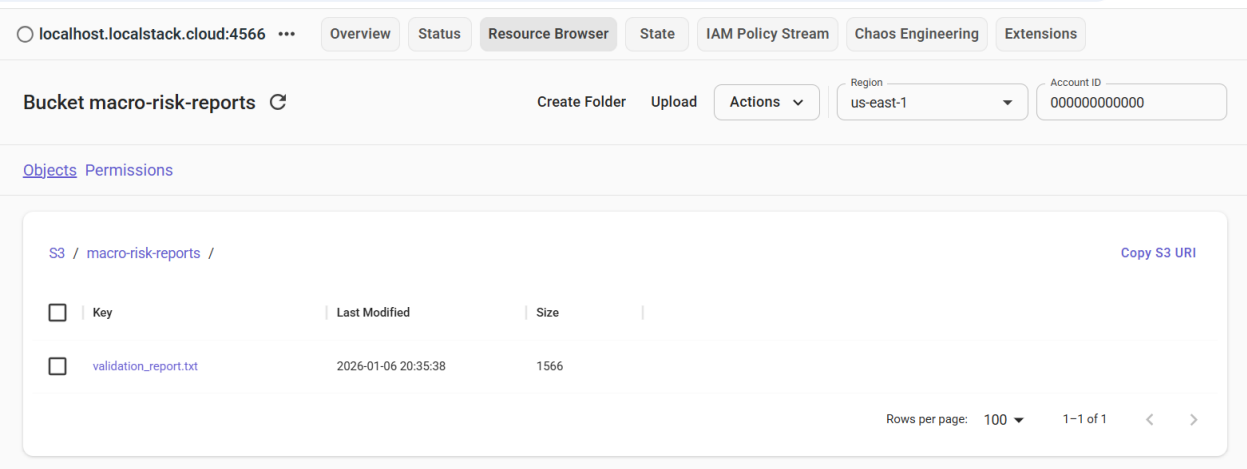
LocalStack Dashboard

For a visual confirmation similar to the AWS Console:

1. Navigate to https://app.localstack.cloud.
2. Select the running instance.
3. Go to Resources > S3.
4. Open bucket macro-risk-reports to view and download the validation_report.txt.

8.3 Cloud API verification

Validate that the API calls function correctly and the bucket/object are physically created.



Main S3 bucket – macro-risk-reports

API status – macro-risk-reports

Service	Operation	Status code	Server time
s3	CreateBucket	✓ 200	2026-01-06 19:44:23
s3	PutObject	✓ 200	2026-01-06 19:44:23
s3	GetObject	✓ 200	2026-01-06 19:47:24
s3	CreateBucket	✓ 200	2026-01-06 19:57:34
s3	PutObject	✓ 200	2026-01-06 19:57:34

S3 operation status – macro-risk-reports