Online Shopping Order Processing System

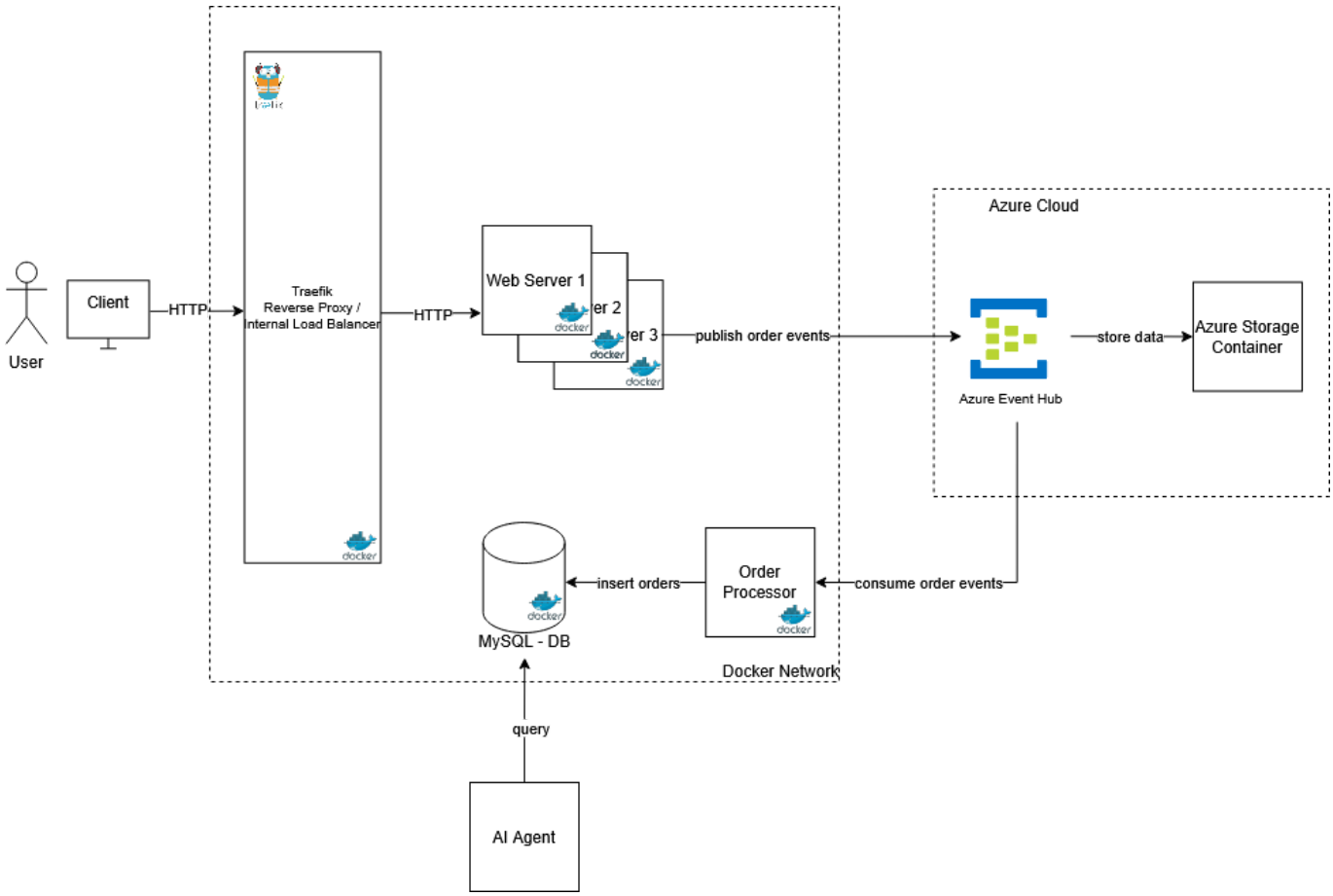
Contributors: Luiz Leão Junior and Susanne Danninger
Link to Repository: https://github.com/susi-dgr/FinalProject-LeaoDanninger
Course: Administração de Redes e Infraestruturas de IT

Project Purpose

The purpose of this project is to build the **infrastructure** for an online shopping order processing system, not the full application logic. It demonstrates a hybrid setup with an on-premise environment using Docker, Traefik, Nginx, NodeJS, and MySQL, combined with cloud services using Azure Event Hub for event streaming and long-term storage.

The infrastructure is managed using **Terraform** for Azure resources and **Ansible** for local automation. An **AI agent** is also included to show how it can interact with the database. The focus of the project is on infrastructure topics such as load balancing, event-driven architecture, automation, and storage, rather than application features.

Order Management System Architecture



Project Components

- **Traefik**: Reverse proxy and load balancer for incoming HTTP requests.
- **Nginx**: Web server serving static content. The web servers can be scaled horizontally.

- **Azure Event Hub**: Cloud-based event streaming platform for ingesting order events.
- **Azure Stoarge Container**: Long-term archive for order events.
- **Order Processor (NodeJS)**: Service that consumes events from Event Hub and processes them into the MySQL database.
- **MySQL Database**: Operational database storing orders.
- **AI Agent**: An AI-powered agent that can query the Orders Database in natural language.

Typical Flow

1. The client makes a request to the Traefik reverse proxy. For the client it appears as a single endpoint, but in reality Traefik load balances requests across multiple Nginx web servers.
2. Traefik routes requests to one of the Nginx web servers for static content. Since this project is mostly to provide the infrastructure and there is no actual web server implementation, the arrow from the web servers to the Azure Event Hub is just illustrative. At this point order events are generated manually to test the setup.
3. Azure Event Hub ingests order events and captures them in a Storage Container for long-term archiving.
4. The Order Processor service consumes events from Event Hub, processes them, and stores the data in the MySQL database.
5. The AI Agent can query the MySQL database in natural language to retrieve order information.

How was the project built

- **On-Prem Infrastructure**: Built using Docker Compose for container orchestration and Ansible for automation. Resulting in a scalable and manageable on-premises setup, that can be deployed with a single command.
- **Azure Cloud Infrastructure**: Provisioned using Terraform, allowing for infrastructure as code and easy management of cloud resources.
- **AI Agent**: Developed using Langflow, enabling natural language queries to the Orders Database.

How to run the project (Ansible + Terraform)

> Some configurations/commands are Windows-specific. Adjust accordingly for Linux/MacOS.

On-Prem Setup with Ansible

1. Build the Ansible runner image:

```
docker build --no-cache -f on-prem/ansible/Dockerfile.ansible -t ansible-
runner:local on-prem/ansible
```

2. Create .env and set the required variables:

```
copy .env.example .env
```

To make it easy, run this and copy it in the right .env file:

```
echo "EVENTHUB_CONNECTION_STRING=$(terraform output -raw
eventhub_connection_string)" >> .env
echo "EVENTHUB_LISTEN_CONNECTION_STRING=$(terraform output -raw
eventhub_listen_connection_string)" >> .env
echo "EVENTHUB_SEND_CONNECTION_STRING=$(terraform output -raw
eventhub_send_connection_string)" >> .env
echo "EVENTHUB_NAME=$(terraform output -raw eventhub_name)" >> .env
echo "EVENTHUB_CONSUMER_GROUP=$(terraform output -raw consumer_group_name)" >>
.env
echo "AZURE_STORAGE_CONNECTION_STRING=$(az storage account show-connection-string
--resource-group resourcegn-oms --name omsarchive12345 --query connectionString -o
tsv)" >> .env
```

3. Run Ansible to build and start the stack:

```
docker run --rm -it `
  -v ${PWD}:/work `
  -w /work/on-prem/ansible `
  -v //var/run/docker.sock:/var/run/docker.sock `
  ansible-runner:local `
  ansible-playbook -i inventory.ini site.yml
```

Azure Event Hub with Terraform

1. Login to Azure CLI and set subscription:

```
az login
```

Show your subscription ID:

```
az account list -o table
```

Set your subscription (replace `<SUBSCRIPTION_ID>` with your own):

```
az account set --subscription <SUBSCRIPTION_ID>
```

Make sure the right location for your resources is set in `variables.tf`. For example `westeurope`:

```
variable "location" {
  description = "Azure region"
  type        = string
```

```
  default     = "westeurope"
}
```

2. Terraform commands to create the Event Hub infrastructure:

```
terraform init
terraform plan
terraform apply
```

How to run manually without Ansible (for development/testing)

On-Prem Setup with Docker Compose

1. Move to the `on-prem/` directory:

```
cd on-prem
```

2. Create .env and set the required variables:

```
copy .env.example .env
```

3. Docker Compose

```
docker compose build
```

4. Start the services (with 3 web instances for load balancing):

```
docker compose up -d --scale web=3
```

5. Stop the services:

```
docker compose down
```

Verify it is working

- Web UI at: `http://localhost/`
- Health at: `http://localhost/api/health`
- Traefik dashboard at: `http://localhost:8080/dashboard`

- Sample Order Created event JSON to send to Event Hub: (this can be tested using the Data Explorer in the Event Hub Instance)

```
{
  "type": "OrderCreated",
  "version": 1,
  "occurredAt": "2026-01-12T12:00:00.000Z",
  "order": {
    "orderId": "ORDER-1099",
    "customerId": "CUST-123",
    "total": 20,
    "items": [
      { "sku": "SKU-001", "qty": 2, "price": 10 },
      { "sku": "SKU-ABC", "qty": 1, "price": 10 }
    ]
  }
}
```

- Check the Azure Storage Container to see if the event was archived.
- After this you can for example use the AI Agent to query the database.

How to run the AI Agent

Langflow Desktop was used to create an AI agent that can query the MySQL database in natural language. To make this work, a few steps are necessary:

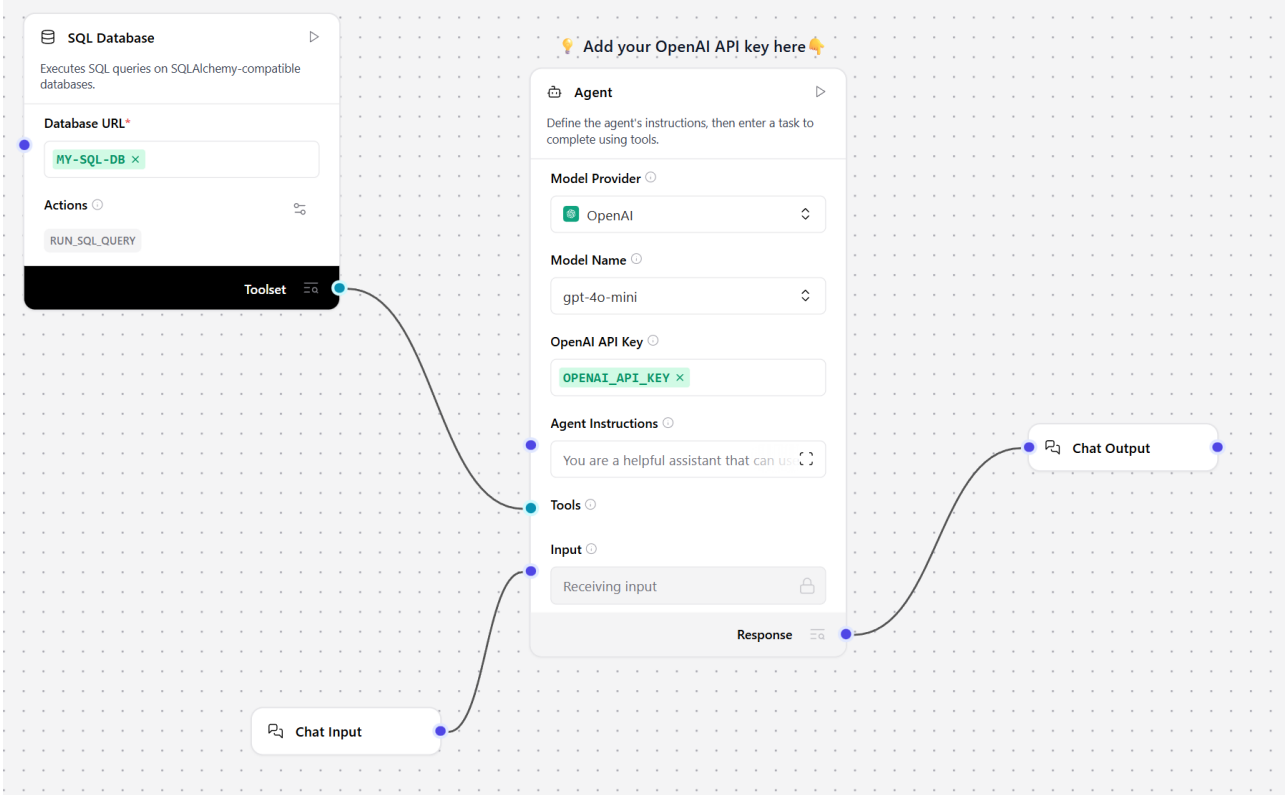
1. Install Langflow
2. Modify `requirements.txt` file found in the directory `C:\Users\USER\AppData\Roaming\com.LangflowDesktop\data` (replace `USER` with your Windows username) to include the following packages:

```
pymysql==1.1.1
```

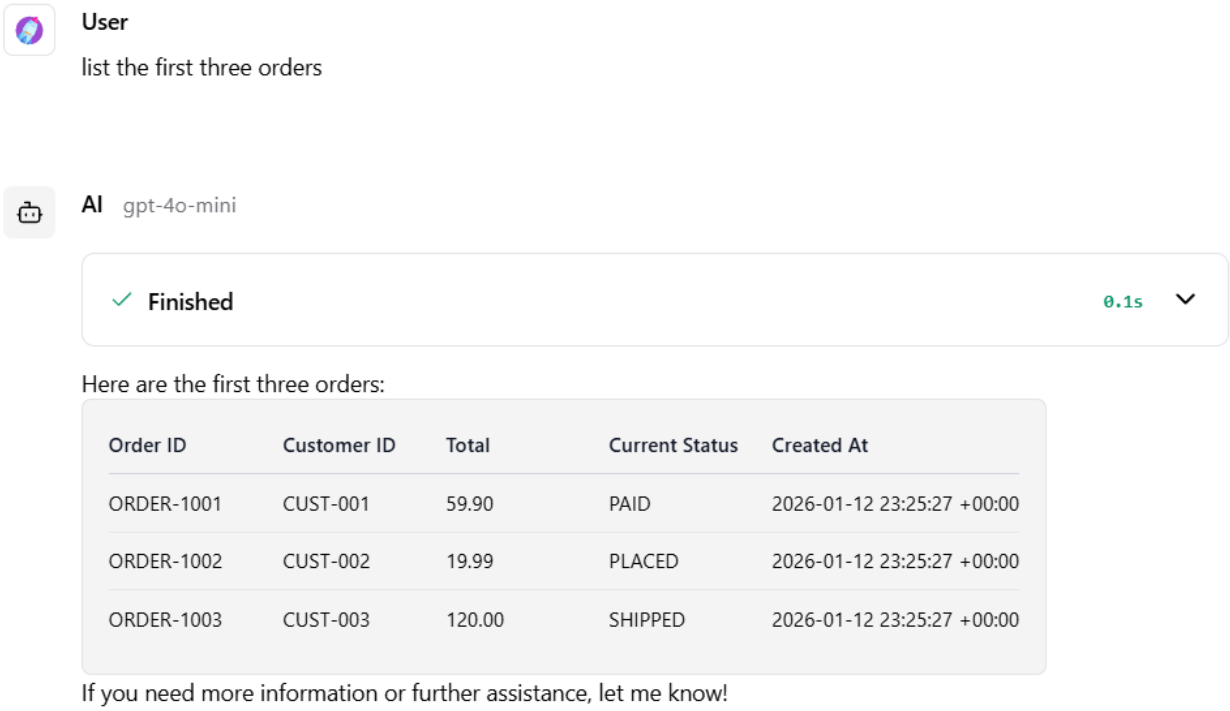
3. Add a global variable in Langflow with the following details:

- Name: `MY-SQL-DB`
- Value: `mysql+pymysql://oms_user:oms_pass@localhost:3306/oms`

4. Setup the AI agent flow as shown below:



5. In the playground you can then query the database in natural language:



Learnings and Challenges

Traefik & Docker Compose

Using Traefik together with Docker Compose was very straightforward. Service routing and load balancing were easy to configure, especially because of prior experience with Traefik. The label-based configuration worked well, and the load-balancing setup behaved exactly as expected. Traefik proved to be powerful while still being easy to get started with.

Terraform & Azure

Terraform with Azure Event Hubs was easier than expected once the Azure subscription was set up correctly. Choosing the right Azure region was important, since not all services are available or enabled everywhere. After understanding how to retrieve values like `EVENTHUB_CONNECTION_STRING` using Terraform outputs and the Azure CLI, the overall workflow became clear and manageable.

Ansible

Ansible was by far the biggest challenge in the project and the part that consumed the most time. It was much harder to understand and implement than expected, with many points where the setup could fail, especially when combined with Docker and different execution environments. Frequent failures and non-obvious error messages made debugging difficult, and the large amount of documentation and online resources was often overwhelming rather than helpful. Despite this, Ansible is clearly very powerful, and mastering it represents the greatest learning opportunity for future projects.

Langflow

Langflow was surprisingly easy to use. Building an AI agent was simpler than anticipated. The main limitation was the lack of a built-in SQL tool, which required extending the setup by adding Python database dependencies. Once this was solved, working with Langflow was smooth and intuitive.