Omni – A Scalable and Fault-tolerant Platform for AI Services

Oisin Mc Menamin

19395331

Ruben Van Breda

19200704

Kamil Michalski

18469806

Dawid Skraba

19433692

**Synopsis:**

We created a project that allows a user to input a prompt, which will then be reproduced and sent to a number of distributed AI services, to return a text or image response to a given prompt. The project has been built using Docker and the AI client services are running on a Kubernetes deployment to ensure fault tolerance and adequate scalability. Having an accumulation of responses from different AI’s can have many benefits, but the purpose of this project is to create the infrastructure to facilitate these calls to distributed services. By doing so, we aim to demonstrate the potential of combining multiple AI sources to generate a diverse and creative set of outputs for the given input.

**Technology Stack:**

*List of the main distribution technologies you will use*

* *Docker : Creating isolated containers*
  + *Container orchestration system, which has been used for service discovery, scaling and load balancing.*
* *Java 8.1*
  + *Java is the primary programming language used for the backend structure.*
  + *Java it’s a good OOP language. The team is familiar with the language. Our broker implementation with Java allows the integration of client services that are written in a variety of different languages.*
  + *Springboot, based on the Spring framework. Used for our client webpage.*
* *Python*
  + *Used for a service bot*
  + *FLASK(REST)*
* *GO*
  + *Used for a service bot*
* *Node JS*
  + *Used for a service bot*
  + *Express (REST)*
* *Maven*
  + *software project management, manage a project's build*
  + *Packaging service for Java*
* *Kubernetes*
  + *Ensures fault tolerance and scalability*
  + *Container orchestration system, which has been used for service discovery, scaling and load balancing*
* *Shell, Batch scripts*

*Highlight why you used the chosen set of technologies and what was it about each technology that made you want to use it.*

We chose to use Docker because it offers the ability to create isolated containers, which help us in maintaining consistent development, staging, and production environments. Docker's container orchestration system has been instrumental in our project for service discovery, scaling, and load balancing. Java 8.1 was selected as the primary programming language for the backend structure because of its robustness and our team's familiarity with it. Java's object-oriented programming (OOP) paradigm allows for organized, modular, and reusable code. Additionally, our broker implementation in Java enabled the integration of client services written in a variety of different languages. Furthermore, we utilized Spring Boot, which is based on the Spring framework, for our client webpage because of its simplicity, versatility, and the productivity it offers in developing stand-alone, production-grade Spring-based applications. Python was used for creating a service bot due to its simplicity, readability, and wide range of libraries. Moreover, we employed Flask, a micro web framework written in Python, to create a RESTful API for the service bot, providing an efficient and standardized way for it to communicate with other services. We also utilized Go language for another service bot due to its simplicity, efficiency, and strong support for concurrent programming. This made it a perfect choice for developing high-performance applications. Node.js was used for yet another service bot, leveraging its event-driven, non-blocking I/O model that makes it lightweight and efficient. We used Express.js, a minimal and flexible Node.js web application framework, to provide a robust set of features for web and mobile applications. Maven was selected for software project management and handling our project's build. Its packaging service for Java allowed us to automate the build lifecycle and manage dependencies effectively. We opted for Kubernetes as it ensures fault tolerance and scalability, which are critical for maintaining high availability and performance of our applications. Like Docker, we leveraged Kubernetes' container orchestration system for service discovery, scaling, and load balancing. Lastly, we employed Shell and Batch scripts for automating repetitive tasks, simplifying complex procedures, and customizing our system operations. These scripting languages provided us with the flexibility and control needed to efficiently manage our development environment.

**System Overview:**

|  |  |
| --- | --- |
| **Module** | **Summary** |
| omni-core | Dependency containing shared classes for other Java modules. |
| omni-broker | Main service that acts as intermediary between user facing clients and the Kubernetes Clusters. It makes requests to all known applicable pods. It then accumulates the response from the different AI’s and sends this data back to the user facing clients. |
| omni-client | Command-Line client interface that can be used to make AI requests via the broker module. |
| omni-client-web | Web based GUI used to interact with AI’s via the broker. Implemented using SpringBoot with HTML, JavaScript, and CSS. |
| omni-integration-davinci | A **Python** service that can return AI generated images or text responses in the style of Davinci. It uses **Flask** to communicate with the Java broker module following the REST architectural Style. |
| omni-integration-picasso | A **Python** service that can return AI generated images or text responses in the style of Pollock. It uses **Flask** to communicate with the Java broker module following the REST architectural Style. |
| omni-integration-pollock | A **NodeJS** service that can return AI generated images or text responses in the style of Poll. It uses **Express** to communicate with the Java broker module following the REST architectural Style. |
| omni-integration-turbo | A **GO** service that can return can give AI generated text responses to prompts. |

Diagram

Description automatically generated

This project creates distributed calls to different AI services, using Kubernetes to provide scalability and fault tolerance. The project follows the SoC (*Separation of Concerns*) principle by separating business logic and user-facing logic through the use of loosely coupled microservices.

The project can be interacted with via a command line service or through the use of an interactive web-based GUI. The broker service interprets inputs produced by these services and creates appropriate requests to discovered distributed AI clients, which each in turn make remote requests to AI services. These clients then send their responses back to the broker which accumulates the data before returning the responses back to the user-facing client. Communication between the broker service and the AI clients, is performed following the REST architecture.

The project is implemented using Docker and a Kubernetes deployment, which can be started by running an included Batch script. Kubernetes provides fault tolerance for the distributed AI clients, and its internal load balancer can scale these AI service containers under times of high traffic.

**Contributions:**

*Provide a sub section for each team member that describes their contribution to the project. Descriptions should be short and to the point.*

Dawid Skraba

* Created the Turbo Bot(text-based) using Go-lang
* Created an image bot called Pollock in Python
* Created the front end of the application
* Modified the application to dynamically allocate addresses of all the bots
* Modified the model so that multiple bots can be added to it dynamically, therefore the model would be updated and the frontend would reflect those new bots being added
* Contributed to debugging and minor programming tasks around the projects classes

Oisin Mc Menamin

* Created the Broker Service.
* Defined how individual bots would work through the implementation of a mock Bot early in development.
* Dockerised the Project.
* Created the command-line client interface.
* Collaborated with team members for debugging, idea generation, and other programming problems.
* General Project and Git Repo management.

Kamil Michalski

* Created the Picasso bot (NodeJS)
* Picasso bot gives prompt responses in text and image form.
* Added the capability of running the project in Kubernetes.
* Added shell/bat scripts for running the project in linux and windows on Docker and Kubernetes.
* Created a demo video for the project.

Ruben Van Breda

* Created the Davinci Bot (Python)
* Added skeleton for springboot web application
* Added image support to the web application, enabling users to upload and display pictures.
* Contributed along with team’s efforts with general programming tasks, such as debugging and troubleshooting. (e.g., added run.bat)
* Collaborated with other team members to ensure the project was completed on schedule and to a high standard.

**Reflections:**

*What were the key challenges you have faced in completing the project? How did you overcome them?*

*One of the key challenges we faced was managing the API keys for different AI services. API keys are codes used to identify and authenticate an application or user. We had to ensure that each AI client had a valid and secure API key to access the remote AI service. We overcame this challenge by using environment variables to store and pass the API keys to the AI clients, and by using encryption and access control mechanisms to protect the API keys from unauthorized use or exposure.*

*API keys were not freely available ( or not open to the public) and required team members to pay in order use and test our project.*

*What would you have done differently if you could start again?*

If I were to start again, one of the things I would consider doing differently is implementing a database for storing responses from AI bots. This could facilitate better tracking, analysis, and potential improvement of their performance. Additionally, I would have cached previous prompts in order to enhance performance. This could potentially lead to faster response times and a smoother user experience. Another change I would consider would be developing a database for tracking the likes and dislikes associated with each model. This would likely be more efficient and sophisticated compared to using a simple text file, and could provide a more in-depth understanding of each model's performance and user preferences. Lastly, it would be beneficial to incorporate more tests into the process. Regular testing can help identify any issues or areas for improvement early on, ensuring the highest quality and reliability of the system.

*What have you learnt about the technologies you have used? Limitations? Benefits?*

One of the critical insights I've gained involves the usage limits of ChatAI. While incredibly powerful and versatile, it is important to acknowledge that like any other technology, it has its limitations which need to be taken into account when designing systems and applications. The models we queried had certain chat limitations, where we had a limit on the amount of requests it could handle, after that point we needed to pay for this service. Furthermore, I've learned that there can be significant performance differences between various models. This knowledge is crucial as it informs the decision on which model to use depending on the specific requirements and objectives of a given project. This also caused for the wait times to differ wildly from one request to another as we had to wait for all models to return a response before showing all responses to the user. In using Kubernetes, I've come to appreciate its easy ability to add fault tolerance and scalability to applications. This feature is highly beneficial, especially for large-scale projects that need to maintain high availability and performance even under demanding circumstances. My experience has also shown me that RESTful architectures are excellent for facilitating communication between services. They provide a simple, standardized way to ensure that different parts of a system can effectively interact with each other. I've been genuinely impressed with the potential of AI technologies. Their ability to automate complex tasks, provide valuable insights, and drive innovation across various fields is truly remarkable. Lastly, I've found that while JavaScript is a versatile and widely-used language, it can be effectively replaced by Thymeleaf implementations in certain contexts. This realization has broadened my understanding of the range of tools available for web development and the specific benefits they can offer.