Agentic Data Platform

Enhancing Data Platform Efficiency with Conversational AI and Automated Configuration Management

Table of Contents

[Table of Contents 2](#_Toc184061336)

[1. Version Control 3](#_Toc184061337)

[2. Concept Overview 4](#_Toc184061338)

[3. Problem Statement 4](#_Toc184061339)

[4. Proposed Solution 5](#_Toc184061340)

[5. System Design 5](#_Toc184061341)

[5.1. Evaluation of Market Solutions 5](#_Toc184061342)

[5.2. Evaluation of Chat UI Frameworks 6](#_Toc184061343)

[5.3. Design Considerations 6](#_Toc184061344)

[5.4. Proposed Approach 6](#_Toc184061345)

[6. Technical Requirements 10](#_Toc184061346)

[7. Workflow Description 10](#_Toc184061347)

[8. Implementation Plan 11](#_Toc184061348)

[9. Proof of Concept (PoC) 11](#_Toc184061349)

[10. Conclusion 11](#_Toc184061350)

[13. References and Appendix 11](#_Toc184061351)

1. Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Autor | Change |
| 0.1 | 08.11.2024 | Eugene Zaidun | Initial Concept |
| 0.2 | 15.11.2024 | Eugene Zaidun, Jan de Bruin | Validation of Concept |
| 0.3 | 22.11.2024 | Eugene Zaidun | Agentic Workflow Components |
| 0.4 | 29.11.2024 | Eugene Zaidun | Conversational Patterns |
| 0.5 | 13.12.2024 | Eugene Zaidun | Workflow Description |

2. Concept Overview

Managing data effectively is crucial for any modern organization, and a unified data platform plays a central role in achieving this. However, configuring data pipelines on platforms like Azure and Databricks often requires significant technical expertise, making it a challenge for many clients. To bridge this gap, we propose creating a user-friendly AI-powered NLP interface that uses Natural Language Processing (NLP) to guide clients through the process of generating configuration files and supporting documentation.  
  
The NLP interface will provide an interactive experience where clients answer a series of questions in a conversational format. Based on their responses, the system will automatically generate a configuration file that complies with organizational standards. This file will be seamlessly integrated into the Git repository to automate the provisioning of data pipelines within the Azure ecosystem. The interface will make the platform more accessible to non-technical clients while maintaining the efficiency and consistency required by the organization.  
  
This solution brings several advantages:  
  
 • **Ease of Use**: clients can set up pipelines without needing technical expertise.  
 • **Time Savings**: Automating the process reduces manual effort and speeds up delivery.  
 • **Consistency**: Standardized configurations ensure high-quality results across the platform.  
  
By introducing this conversational approach, the data platform can empower clients across the organization, making data management more inclusive and efficient.

3. Problem Statement

The RIVM’s data platform, though newly deployed and recently transitioned from testing to a production like environment, is already demonstrating its potential as a central hub for managing data. However, as the team works to stabilize the system, refine configurations, and tackle technical challenges, it faces a unique set of obstacles that must be addressed to ensure the platform’s success and scalability.

|  |  |
| --- | --- |
| Challenge | Description |
| Complexity of Configuration | Setting up metadata configurations requires advanced expertise not all users possess. |
| Integration of External Data Sources | Incorporating external data seamlessly introduces new technical challenges. |
| Limited Accessibility for Non-Technical Users | Non-technical users struggle with navigating the platform’s tools. |
| Ongoing Development Demands | Limited capacity to address user-facing challenges due to active development and system stabilization efforts. |
| Manual Documentation Management | New clients require detailed documentation for their data pipeline configurations, which currently must be managed manually, adding to the workload and slowing the onboarding process. |

These issues collectively hinder the platform’s ability to be fully adopted by the organization, reduce its efficiency, and limit its impact. Addressing these challenges is essential to ensure that the platform evolves into a scalable, accessible, and efficient tool for managing data across the organization.

4. Proposed Solution

We propose building an AI-powered NLP interface to tackle two key issues: the complexity of creating Databricks configuration files and the manual effort needed to produce documentation for data pipelines. This tool will make it easier for clients to set up configurations and ensure every pipeline comes with clear, client-ready documentation.  
  
The NLP interface will guide clients step-by-step through a conversation to gather all the necessary details. Based on this input, it will automatically generate configuration files that meet organizational standards and create detailed documentation explaining how the pipeline works, how it connects to external systems, and how clients can use it. This will save time and reduce the workload for the platform team.  
  
This solution directly targets the two main problems:  
  
 • **Configuration Complexity**: Simplifying the creation of Databricks configuration files to reduce technical barriers for clients.  
 • **Documentation Automation**: Simplifying the process of generating accurate, client-ready documentation, ensuring consistency and efficiency.  
  
By integrating these capabilities into a unified interface, the solution will enhance user accessibility, save time, and improve the overall scalability of the platform.

5. System Design

Building a system capable of simplifying complex configurations and automating documentation requires careful consideration of capabilities and design complexity. Such a system must enable rich conversations with end-users to achieve specific goals, provide an intuitive interface for interaction, support user task-related activities, and execute code within the platform’s ecosystem. These requirements make the solution complex and demand a robust and scalable design.

5.1. Evaluation of Market Solutions

To address these needs, we began by exploring existing solutions in the market, evaluating their capabilities, and assessing their feasibility for implementation:  
  
1. **Microsoft Compiler 365 Suite**:  
  
 • This suite offers a rich ecosystem with AI Agent Builder, low-code/no-code capabilities, and seamless integration options. It was initially considered as an option for building the conversational AI system.  
 • However, after consulting with the Innovation Department’s specialist manager, we determined that the licensing cost of over $100,000 for organization-wide deployment made it financially impractical.  
  
2. **Community-Based Frameworks**(given the high cost of proprietary solutions, we shifted our focus to leveraging community-available frameworks to implement the required system. These frameworks offer flexibility and avoid the heavy licensing burden). Potential options include:  
 • Autogen by Microsoft  
 • LangGraph by LangChain  
 • Swarm by OpenAI

5.2. Evaluation of Chat UI Frameworks

To simplify the development of the chat interface, we evaluated the availability of frameworks and tools in the market designed specifically for building user-facing conversational applications. Prominent vendors such as Hugging Face and OpenAI have developed advanced ecosystems tailored for creating such applications. These ecosystems toolkits that simplify the integration of conversational capabilities into user interfaces.

5.3. Design Considerations

Implementing such a solution requires addressing several critical factors:  
 • Runtime and Ecosystem: The chosen framework must support runtime execution of user tasks and background code within the platform.  
 • Deployment: The system must operate within a manageable and scalable environment, likely requiring a dedicated ecosystem or “landing zone” to host its components.  
 • Development Effort: Unlike off-the-shelf solutions, a community-based framework means the system must be built from scratch, demanding significant development resources and expertise.  
 • Scalability and Flexibility: The solution must be able to evolve with the platform, accommodating new features and use cases over time.

5.4. Proposed Approach

Given the evaluation and design considerations, we propose leveraging a Proof of Concept (POC) landing zone within the RIVM Azure ecosystem. This approach simplifies management and reduces the need for security considerations, as full responsibility for deployments in this environment rests with the development team. This setup allows for faster iteration and experimentation during the initial phases.  
  
For further implementation, we recommend adopting containerization for application packaging. This approach ensures portability, scalability, and ease of deployment across various environments, aligning with best practices for modern application development.  
  
Additionally, we propose implementing a multi-tier architecture for the application. This design pattern will separate concerns such as the user interface, business logic, and data access layers, making the system easier to manage, maintain, and scale over time.  
  
Finally, for accessing foundation model APIs, we suggest utilizing Azure Cognitive Services. This service provides robust, enterprise-grade support for AI capabilities, ensuring seamless integration with the existing Azure ecosystem while minimizing development overhead for foundational AI components.

|  |
| --- |
| Multi-Tier Architecture |
|  |

|  |
| --- |
| Backend Service Components |
|  |

|  |
| --- |
| Chat UI Frameworks |
|  |

In the previous three images, we illustrate the concept of multi-tier replication. The backend service diagram highlights the decomposition of core components essential for the implementation. For the chat UI front-end, we’ve detailed the key components alongside themes, treating it as a distinct layer within the architecture.

|  |
| --- |
| Agentic Workflow |
|  |

An agentic workflow is a design pattern for building complex AI applications by orchestrating multiple AI agents, each configured with unique prompt engineering settings and equipped with tools that leverage foundational capabilities of large language models (LLMs), such as function calling. This workflow allows for the creation of systems where agents perform specific tasks independently yet collaboratively, driven by their configurations and responsibilities.  
  
At the core of the agentic workflow is the conversational pattern, which defines how an agent interacts and takes responsibility during task execution. This pattern governs the agent’s behavior, including decision-making and task delegation, and ensures that the workflow remains dynamic and adaptable. The conversational pattern is not static—it can be configured within the agentic orchestration flow to optimize performance for different use cases, enabling execution of complex processes through natural language-driven interactions.

Agentic Workflows Core Components:

|  |  |
| --- | --- |
| Component | Description |
| Agent Runtime | Operates in both single-threaded and distributed environments to ensure flexibility and scalability. |
| Event-Driven Principle | Facilitates responsive and asynchronous communication through a messaging system. |
| Prompt Engineering | Enhances agent capabilities by designing configuration prompts that define behavior and interaction protocols. |
| Integration with External Resources | Connects with external systems such as vector databases and third-party services to extend functionality. |
| Telemetry and Logging | Provides essential monitoring pillars to track performance, detect issues, and ensure reliability. |

Conversational Patterns:

|  |  |
| --- | --- |
| Pattern | Description |
| Group Chat | The group chat design pattern involves a group of specialized agents sharing a common thread of messages by subscribing and publishing to the same topic. Each agent is assigned a specific task, such as writing, illustrating, or editing, for a collaborative project. A human user agent can also participate to provide guidance when needed. |
| Handoff | The handoff design pattern allows agents to delegate tasks to other agents using a special tool call. This pattern, introduced by OpenAI in the experimental Swarm project, facilitates collaborative problem-solving by enabling seamless task delegation among agents. |
| Mixture of Agents | The Mixture of Agents design pattern is inspired by the structure of feed-forward neural networks. It employs multiple layers of worker agents coordinated by a single orchestrator agent, with a focus on task processing in stages. |
| Multi-Agent Debate | The Multi-Agent Debate design pattern enables collaborative problem-solving through multi-turn interactions among agents. Agents exchange, refine, and debate their responses over a series of rounds, culminating in a consensus-based final answer. |
| Reflect | The Reflection design pattern involves a two-step iterative process where one agent generates output (e.g., code), and another agent critiques or reflects on that output. This cycle continues until a stopping condition is met, fostering iterative refinement. |

6. Technical Requirements

TBD.

7. Workflow Description

**Step 1**: Client Interaction with NLP UI interface  
  
 1. The client accesses the system through a client-facing NLP UI interface.  
 2. The system initializes a conversational session and presents the context for the current flow, which outlines the steps the client will go through. One of the key context items is fulfilling the schema requirements.  
 3. The client is guided through a series of steps, starting with questions designed to gather the necessary schema inputs.  
  
**Step 2**: Schema Fulfillment Process  
  
 1. Schema Questionary context:  
 • A dedicated context item is responsible for serving the schema-based questionary functionality.  
 • The system dynamically presents schema requirements as questions, ensuring all necessary data fields are addressed.  
 • Example: If the schema requires “data source type” and “pipeline parameters,” the system asks targeted questions to gather this information.  
 2. Client responses are validated in real-time to ensure they meet the schema’s constraints (e.g., required fields, valid formats).  
 3. For nested or conditional fields, the conversation flow adjusts automatically to capture dependent data.  
  
**Step 3**: Agentic Workflow Execution  
  
 1. Once the schema fulfillment context is completed, the system initiates the agentic workflow:  
 • Configuration Agent: Processes the schema inputs to generate the Databricks configuration file.  
 • Documentation Agent: Creates detailed documentation, explaining the pipeline’s design, data integration points, and operational details.  
 2. Each agent operates independently but collaboratively, leveraging LLM capabilities via Azure Cognitive Services APIs to perform tasks efficiently.  
  
**Step 4**: Background Code Execution  
  
 1. The generated configuration file and documentation are validated against organizational standards.  
 2. Any errors or inconsistencies are logged and communicated to the client via the NLP UI for correction.  
  
**Step 5**: Output Generation  
  
 1. Upon successful validation:  
 • The finalized Databricks configuration file is prepared for deployment.  
 • Detailed, client-ready documentation is generated and shared with the client.  
 2. Outputs are displayed in the chat interface for review and approval.  
  
**Step 6**: Integration with Git Repository  
  
 1. The system commits the configuration file to the designated branch of the Git repository.  
 2. A CI/CD pipeline is triggered to deploy the pipeline to production.  
 3. Logs and version history are maintained for tracking and auditing purposes.  
  
**Step 7**: Feedback Loop and Adjustments  
  
 1. Clients can review the generated outputs and provide feedback directly through the chat interface.  
 2. The system enables iterative refinements, restarting the schema fulfillment context if required, to adjust inputs or outputs.

8. Implementation Plan

TBD.

9. Proof of Concept (PoC)

TBD.

10. Conclusion

TBD.

13. References and Appendix

TBD.