Performance efficiency for SAGA API Orchestrator

Software Architecture
Vivieb

Debayan, Kabita, Kiran, Mhreteabe
Università degli studi dell'Aquila, Italy

Academic year: 2023–2024

1 Introduction

In an era where digital transformation is pivotal, the performance and efficiency of APIs play a crucial role in determining the success of cloud-native applications and microservices. The SAGA API Orchestrator emerges as a cornerstone in this landscape, orchestrating the complex interplay of microservices. However, as the scale and complexity of cloud-native applications escalate, it becomes imperative to ensure that the SAGA API Orchestrator operates at its peak efficiency. This project, centered around enhancing the performance efficiency of the SAGA API Orchestrator within a Kubernetes environment, seeks to address the intricate challenges posed by the burgeoning volume of APIs and the colossal data they manage.

The core of microservices design in today's distributed computing environment is API orchestration, especially when it comes to cloud-native application deployments. A key component of this ecosystem is the SAGA API Orchestrator, which facilitates smooth communication and transaction handling amongst loosely linked services. However, controlling performance efficiency becomes more difficult due to the distributed structure of these services.

The project relies on a robust technology stack, featuring Kubernetes as the microservices hosting platform, complemented by Kube-logging, Elasticsearch, Docker, NodeJS, NextJS, D3JS, SAGA Pattern, and MaterialUI. The data structures are organized using a multi-tenant approach, with MongoDB serving as the reference NoSQL database. The key data structures include Saga, SagaInstance, SagaInstanceData, and SagaInstanceLog, each playing a crucial role in cataloging processes, tracking individual process executions, managing associated data, and recording log information.

The project's ultimate goal is to empower business and technical stakeholders with actionable insights to improve process efficiency. Additionally, the information derived from the analysis will be utilized by external programs to prevent errors and offer valuable suggestions to process consumers. The main dashboard provides a high-level overview of key performance indicators, while detailed dashboards for the process catalog and individual processes offer granular insights, status indicators, and improvement suggestions. Together, these elements form a comprehensive approach to enhancing the performance efficiency of the SAGA API orchestrator in a cloud-native ecosystem.

2 Objective

The primary objective of the project is a comprehensive analysis of execution data within the SAGA API orchestrator framework. Important metrics including pathways, outcomes, standard deviation time, and controlled data are included in this research. Through comprehensive analysis of these data points, the initiative seeks to reveal insights into process performance, finding critical parameters that may be used to improve efficiency.

Predictive analytics are also incorporated into the project to identify processes that may encounter errors or cause delays in completion. After that, the compiled data is arranged in a catalog of processes, providing aggregated views and allowing for a detailed examination of each process execution.

The objectives of this project are discussed below:

1. Analyze execution data from the SAGA API orchestrator, including durations, results, standard deviation time, paths, and managed data.

2. Identify performance metrics for processes, providing insights to enhance their efficiency.
3. Conduct predictive analyses to anticipate potential errors or deviations in completion time for specific types of executions.
4. Correlate obtained data within the catalog of processes, creating aggregated views for comprehensive insights.
5. Enable navigation of individual process executions within the catalog, offering a detailed understanding of each process.
6. Provide business and technical stakeholders with actionable information to improve overall process efficiency.
7. Utilize metrics to understand and highlight processes that may benefit from optimization or corrective actions.

3 Requirement Analysis

The requirement analysis phase is critical for shaping the SAGA API orchestrator dashboard. It focuses on understanding project objectives, addressing stakeholder needs, and defining functional, non-functional, and reporting requirements. The key components include:

1. **Functional Requirements**
   (a) **Performance Metrics Analysis** Analysis mechanisms to evaluate and present performance metrics for each process, highlighting areas for improvement.
   (b) **Predictive Analysis** It is required to develop predictive analysis capabilities to identify processes at risk of error or likely to exceed the average completion time for their type.
   (c) **Catalog Integration** Correlate and integrate obtained data within the catalogue of processes, providing aggregated views accessible through the main dashboard.
   (d) **Individual Process Navigation** Enable users to navigate through individual process executions within the catalogue, allowing detailed examination of each process's performance.
   (e) **Stakeholder Access** Provide secure access to business and technical stakeholders, ensuring they can utilize the information to enhance process efficiency.
2. **Non-Functional Requirements**
   (a) **Performance** The system must provide real-time or near-real-time data updates and analyses to ensure timely decision-making.
   (b) **Scalability** Design the system to handle a growing volume of data and users, ensuring scalability to meet future demands.
   (c) **Security** Implement robust security measures to safeguard sensitive execution data and ensure only authorized users can access the system.
   (d) **User Interface** Design an intuitive and user-friendly interface, enabling stakeholders with varying technical backgrounds to interpret and utilize the dashboard effectively.
   (e) **Compatibility** Ensure compatibility with various browsers and devices, allowing stakeholders flexibility in accessing the dashboard.

4 Challenges/Risk Analysis

This project's success depends on how well possible risks and problems are managed. A thorough grasp of the intricacies of real-time analytics, scalability issues, and integration challenges is necessary for success. It is imperative for project resilience, sustained performance, and alignment with overall goals to practice proactive management, which is the methodical identification and mitigation of these obstacles. This section will provide a comprehensive analysis of the challenges and potential risks associated with the implementation of the project. The project entails creating and deploying a dashboard system with multiple indicators to monitor and manage process execution across the platform.

1. **Challenges Identified**

   (a) **Integration Complexity** One of the primary challenges lies in the integration of diverse processes within the platform. Each process has its unique characteristics, and ensuring seamless integration with the main dashboard requires meticulous planning and execution.

   (b) **Real-time Data Synchronization** The real-time nature of the dashboard, especially in displaying executions in the last hour, poses a challenge in terms of data synchronization and accuracy. Maintaining the timeliness of data across various processes demands efficient data handling mechanisms.

   (c) **Performance Optimization** Ensuring the optimal performance of processes, as reflected in the status indicators on the Process Catalog dashboard, requires continuous monitoring and potential optimizations. Addressing performance issues promptly is crucial to maintaining the overall health of the platform.

   (d) **User Experience** Providing an intuitive and informative user experience is critical for the success of the dashboard. Ensuring that users can easily interpret status indicators, access specific dashboards, and receive actionable insights is a challenge that requires careful design considerations.

   (e) **File structure** The risk involves managing a complex file structure, particularly within nested JSON data. The intricacies of navigating through layers of nested information may pose challenges in data retrieval, processing, and overall system efficiency.

2. **Risk Mitigation Strategies**

   (a) **Thorough Testing** To address integration complexities and potential errors, a comprehensive testing strategy will be implemented. This includes unit testing for individual processes and end-to-end testing to ensure smooth interactions between processes and the main dashboard.

   (b) **Robust Data Management** Implementing robust data management practices will be essential for real-time synchronization. Regular data quality checks and mechanisms for handling discrepancies will be established to enhance the reliability of the displayed information.

   (c) **Continuous Performance Monitoring** To proactively address performance-related challenges, a continuous monitoring system will be implemented. Automated alerts will notify administrators of any deviations from expected execution times, allowing for prompt investigation and optimization.

   (d) **Dashboard User Adoption** Providing comprehensive training sessions for users and emphasizing the value the dashboard brings to their roles. Gathering user feedback regularly to make iterative improvements to the dashboard's usability.

   (e) **Flattening approach** In response to the risk associated with a complex nested JSON file structure, the project will implement a flattening approach. This involves restructuring the nested layers into a more straightforward, flattened format. This strategy facilitates a streamlined approach to handling data, reducing complexities and ensuring optimal system performance.

5 State of the Art Analysis (SoTA)

A state-of-the-art analysis for the described project involves evaluating the current landscape of similar technologies, methodologies, and tools in the field of process management, dashboard creation, and performance monitoring. The goal is to understand the latest advancements and innovations that can enhance the project's features, usability, and overall effectiveness.

This custom solution, developed particularly for Vivieb Company, represents an innovative step forward in process automation and monitoring. Its uniqueness stems from the adoption of a customised dashboard system, making direct comparisons with current projects challenging. The project is custom-designed to fulfill Vivieb's specific requirements, distinguishing it from typical alternatives. It sets a new benchmark for customized business intelligence by focusing on industry-specific key performance indicators and insights. The project's unique approach recognizes its sole focus on meeting Vivieb's needs, positioning it as a forerunner in establishing norms for industry-specific process automation and monitoring solutions.

6 Requirements Refinement

The success of any project hinges on the clarity and precision of its requirements. In the case of this project, involving the creation of a dynamic dashboard with multiple functionalities, the process of requirement refinement is paramount to aligning the project outcomes with the stakeholders' expectations. This section outlines the steps taken in requirement refinement to ensure a robust foundation for the development and implementation phases.

1. **Stakeholder Collaboration** The initial set of requirements was gathered through extensive collaboration with stakeholders, including project managers, and end-users. Through interviews and feedback sessions, we aimed to capture a comprehensive understanding of the objectives and expectations.

2. **Requirement Categorization** The gathered requirements were categorized into distinct modules to facilitate a structured approach to development. This categorization allowed for a clear delineation of features, ensuring that each aspect of the dashboard was addressed systematically

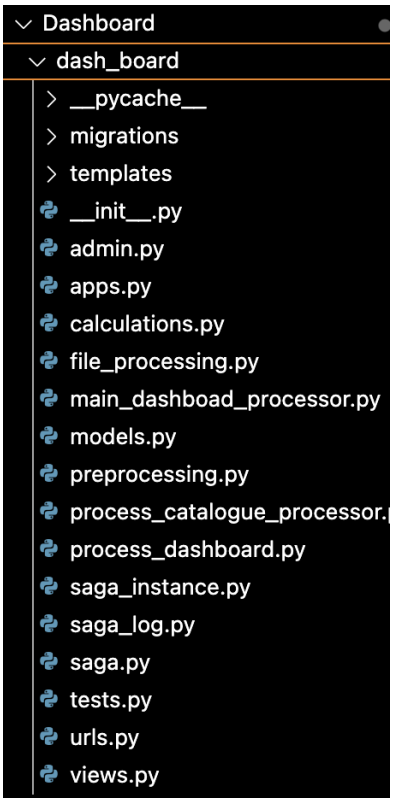

Figure 1: Dashboard components

3. **Technical Feasibility Assessment** Simultaneously, a technical feasibility assessment was conducted to ensure that the identified requirements were achievable within the project's constraints. This involved evaluating the scalability, compatibility, and integration aspects of the proposed features.

4. **Documentation** Comprehensive documentation of the refined requirements, stakeholder's requirements acceptance criteria, and technical specifications was maintained.This documentation served as a reference point for all team members and stakeholders, fostering a shared understanding of the project scope.

7 Architecture of the System

The architecture of the system adopts a Model-View-Template (MVT) architecture using Django, with a primary emphasis on the View Layer as the foundational component. This meticulous organization aims to enhance

modularity, scalability, and maintainability, particularly suitable for managing the complexities of the described system [1].

The View Layer takes a central role as the main component of the architecture, consisting of a view file that orchestrates all logic, data processing, and calculations. This layer is responsible for refining and organizing the processed data, setting the stage for downstream processing. The detailed architectural view can be seen in the figure 2.

Following the View Layer, the Model Layer executes queries and manages the data received from the SAGA API orchestrator. Model files within this layer handle initial data processing, ensuring it is ready for subsequent analysis. The Model Layer plays a vital role in data management and contributes to the overall efficiency of the system.

Above it, the Template Layer comprises a set of HTML pages, including the dashboard, for presenting the processed data to users. These templates focus on rendering the data in a user-friendly manner and facilitating user interaction, guided by the actions and decisions orchestrated by the View Layer.

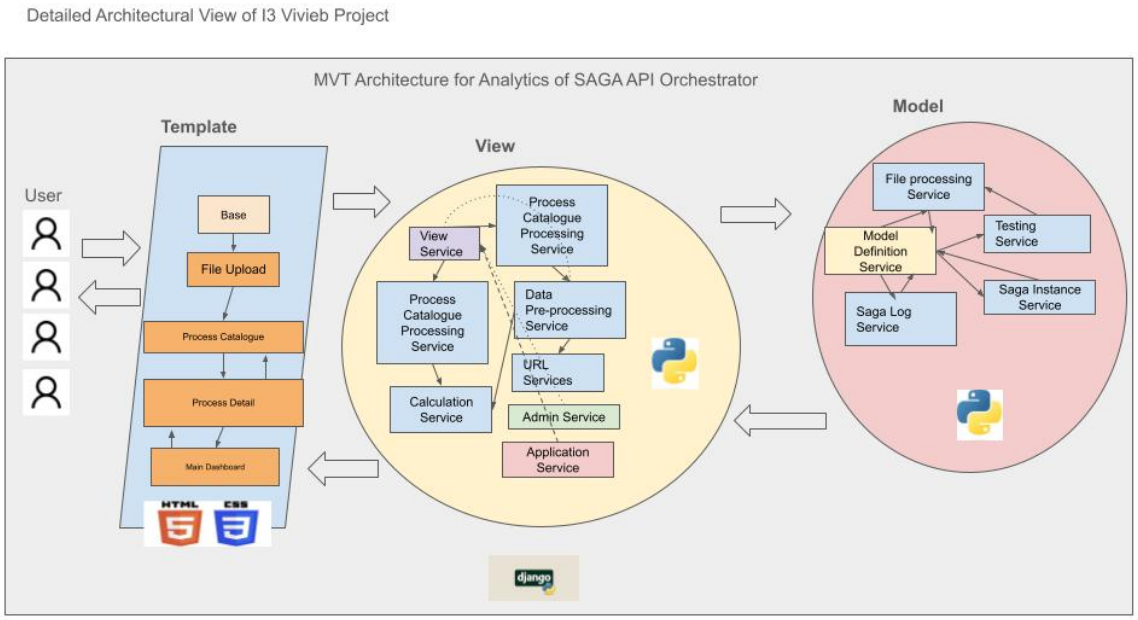

Figure 2: Detailed Architecture View

1. **Pros of Model-View-Template (MVT) Architecture:**
   (a) **Modularity and Maintainability:** The structured organization ensures that the View Layer, as the central orchestrator, is responsible for a specific aspect of the application. This facilitates ease of maintenance and updates without impacting the entire system.
   (b) **Scalability:** The View Layer can be independently scaled based on specific processing demands, providing flexibility and efficiency in resource allocation.
   (c) **Isolation of Concerns:** The View Layer is designed to be self-contained, fostering clear separation of concerns. This isolation enhances fault isolation and simplifies debugging and troubleshooting.
   (d) **Technology Stack Flexibility:** Different layers or components, including the View Layer, can leverage distinct technologies, enabling the use of the most suitable tools for specific tasks.

1. **Cons and Challenges:**
   (a) **Increased Complexity:** The MVT architecture introduces additional complexity due to the interdependencies between layers and the need for effective communication between components.
   (b) **Potential Performance Overhead:** Communication between layers can result in performance overhead, particularly if not managed efficiently.
   (c) **Deployment Challenges:** Coordinating the deployment of multiple layers and ensuring seamless integration can be challenging, necessitating careful planning and execution.

(d) **Potential for Tight Coupling:** Inadequate design decisions may lead to tight coupling between layers, compromising the desired modularity.

In conclusion, the Model-View-Template (MVT) architecture adopted in our system places a significant focus on the View Layer, considering it as the foundational and central component. This approach offers a scalable, modular, and maintainable framework for analyzing execution data. The careful organization of layers and components, with the View Layer as the main orchestrator, facilitates efficient data processing and analysis, providing valuable insights into the performance and outcomes of the SAGA API orchestrator's executions.

8 Design Decisions

Design decisions in software architecture are pivotal choices made during system design, determining the structure and behavior of a software application. Crucial for clarity, these decisions ensure a consistent understanding among team members and facilitate systematic problem-solving. They involve trade-offs, addressing conflicting goals, and guiding developers in navigating constraints. Well-made decisions contribute to flexibility, adaptability, and risk mitigation, anticipating future changes and aligning with project requirements [2].

The architectural design decisions for this project have been thoughtfully and deliberately formulated, prioritizing the creation of a system that is not only robust but also scalable and adaptable to the dynamic requirements of modern microservices orchestration. These key design choices encapsulate the project's core principles and objectives

1. **Modular-View-Templates (MVT) Architecture:** A significant design decision was the adoption of the Modular-View-Templates (MVT) architecture. This approach delineates distinct layers, each with a specific role, akin to the principles of MVT. The decision to employ MVT ensures a well-organized system where updates or fixes to one layer can be implemented without affecting others. This architecture enhances organizational clarity and provides a structured framework for efficient development and maintenance, aligning with the principles of Modular-View-Templates for a streamlined and adaptable system design.

2. **Python as the Core Language:** We went with Python as the main language for our project. Python can do a lot of things and is easy to understand. It helps us write code quickly, and its versatility allows us to handle various tasks seamlessly.

3. **Django Framework:** For building the web part of our project, we chose Django. It's a toolkit that helps us create websites fast and efficiently. Django takes care of a lot of things behind the scenes, so we can focus on making our dashboard look good and work well.

4. **HTML and CSS for the User Interface:** To make our dashboard visually appealing and user-friendly, we used HTML and CSS as can be seen in 3 HTML is like the structure of a house, defining how things are laid out. CSS is the paint and decorations, making everything look nice. Together, they create a dashboard that's easy on the eyes and easy to use.

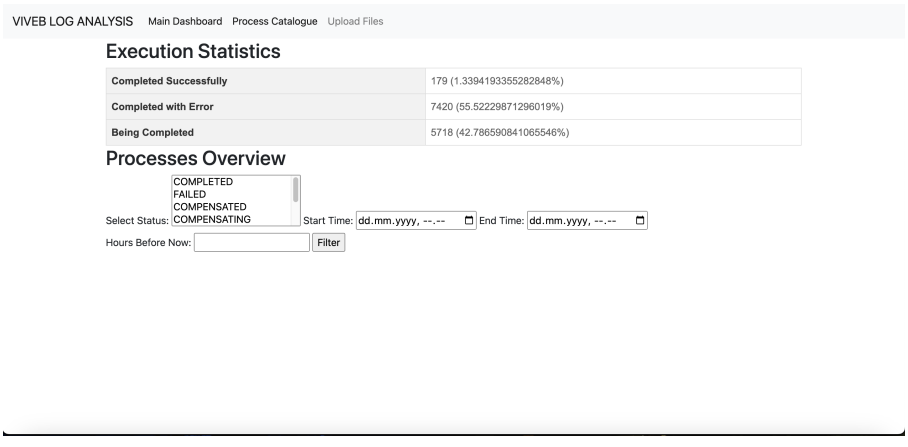

Figure 3: Main Dashboard

5. **Modules for Specific Tasks:** Organizing our project into task-specific modules is a strategic choice. Each module acts like a specialist, efficiently managing a specific aspect. These modules collaborate seamlessly, contributing collectively to project success. This modular approach enhances flexibility and scalability, enabling targeted updates to individual modules without system-wide disruptions. Adopting modules aligns with our goal of a well-coordinated, flexible, and scalable system architecture, promoting efficient collaboration among specialized components.

6. **Modular Data Processing:** We decided to process data step by step, starting from the Data Ingestion Layer, going through Data Flat, Processing, and Calculation Layers, and finally reaching the Results Layer. This modular approach makes it easier to understand and manage each part of the process.

7. **User-Centric Dashboard Design:** When designing the dashboard in the Result Layer, we kept the end-users in mind. We used HTML and CSS to create a user interface that's not just functional but also looks good. As it can be seen in the figure 4 and figure 5. A user-friendly design ensures that people can easily interact with and understand the insights provided by the dashboard.

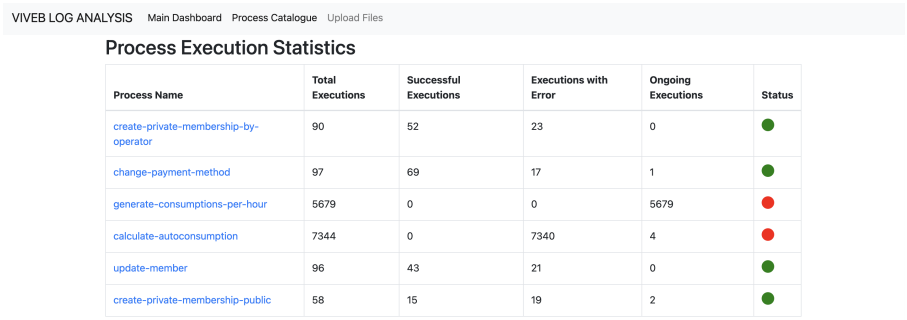

Figure 4: Process Catalogue

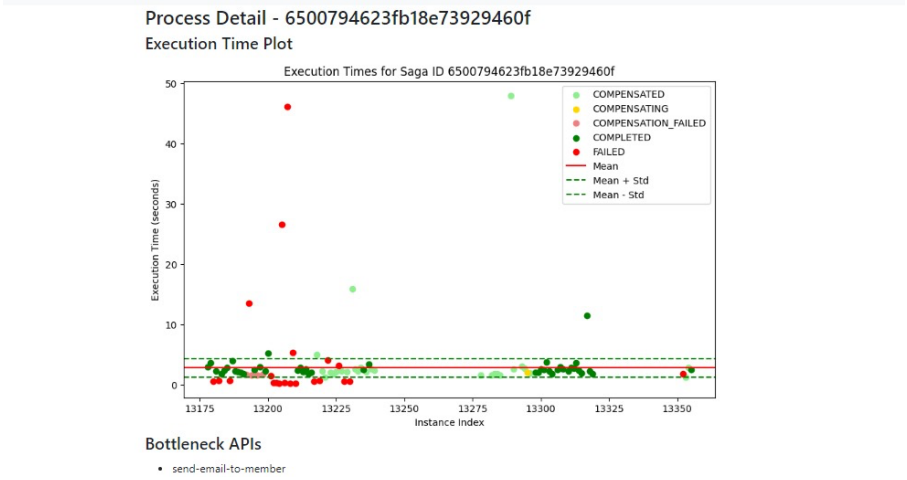

Figure 5: Process Details

9 Summary

The Performance Efficiency for SAGA API Orchestrator Project aims to optimize microservices orchestration within Kubernetes by utilizing a MVT (Module-View-Template) marchitecture. The major focus is on five essential components: the Data Analysis Module, Catalog Management System, and User Interface.

The Data Analysis Module is at the heart of the system, using algorithms to calculate performance indicators and anticipate future outcomes. The Catalog Management System ensures that data is structured and correlated, and the User Interface provides stakeholders with intuitive dashboards. External Program Integration improves accessibility, while the Suggestion Engine generates actionable insights.

With a deliberate adoption of an MVT marchitecture, the project is strategically positioned to deliver a resilient solution. By excluding SAGA API Orchestrator and Microservices from the refined architectural focus, the project emphasizes adaptability and scalability. This approach underscores a commitment to empowering stakeholders with comprehensive insights for informed decision-making and continuous process optimization.

References

[1] Rick Kazman, Mark Klein, and Frank Bachmann. *Software Architecture in Practice*. Addison-Wesley, 3rd edition, 2007.

[2] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120, 2005.