Test Plan – eComposter Cloud Solution

Bruno C. Marques - 645664

Alexander Arkhipov - 647833

Sander Harks - 650397

Chung Hey Lam - 639171

BSc Information Technology

Inholland Hogeschool

Cloud Computing

21 Jan 2024

Table of contents

Table of contents **2**
Context of the product **3**
Value of the product **4**
Requirements **5**
Testing methods **7**
Linting 7
Unit testing 7
Integration testing 8
End 2 end testing 9
Risks and mitigations **10**
Security 10
Outages 10
Performance 11
Stability of the production environment during testing 12
Automation and deployment **12**
GitHub Actions 12
Bicep templates & deployment scripts 13
Environmental isolation 14
Observability **14**
Conclusion and Advice **16**

Context of the product

Mago Bio Solutions operates within the sector of sustainable waste management. This sector is focusing on innovative solutions for recycling and transforming organic waste into valuable resources. Mago has positioned its eComposter machines as a pivotal technology in this domain, targeting a diverse clientele that includes hotels, restaurants, and bars. However, the company's growth is currently blocked by operational inefficiencies due to the lack of a management platform for these machines.

Mago's existing system relies on manual interventions for managing the eComposter machines, making it challenging to grow their company. The lack of a platform for machine management has created a bottleneck in operational workflow and client interaction. Recognizing this issue, Mago started the project to develop a cloud-based solution that functions independently with the existing IoT infrastructure.

The project was made to address the specific needs of Mago and its clients. The primary objective is to develop a cloud-based back-end solution that facilitates direct communication and data transfer from the eComposter machines. This solution aims to provide a seamless, scalable, and efficient solution that minimises manual interventions and maximises user experience.

The solution is made to use cloud technology, APIs, and microservices to establish a robust and flexible back-end system. This system is designed to independently handle data flow, machine management, and client interactions, hopefully enhancing Mago's operational capacity.

Developing a cloud solution in the sustainable waste management sector involves unique challenges, including integrating with diverse IoT devices, ensuring data security, and maintaining system robustness amidst varying client demands. The solution must be adaptable to different scales of operation, from small restaurants to large hotel chains, ensuring consistent performance and reliability.

Value of the product

The cloud-based solution is designed to make Mago's operations more efficient. By automating data transfer and machine management processes, it reduces the need for manual interventions, allowing Mago to focus on other tasks. This efficiency is important for scaling up the business and handling a growing client base without an increase in operational complexity or costs.

One of the tasks of the cloud solution is giving Mago's clients the possibility to manage their eComposter machines. This autonomy not only increases client satisfaction but also reduces the workload on Mago's engineering team, improving operational efficiency even more.

The cloud architecture makes sure that the system can easily scale whenever there is an increase of clients and machines. Its design allows for easy integration with different IoT devices and makes sure to adapt to the evolving needs of Mago and its clients, which is important to make sure there is room for growth.

The cloud solution also offers Mago a huge amount of data such as insights of the machine performance, usage patterns and any problems that occur with the machines. This information is important to make certain decisions when changing settings of the machine, such as why is the temperature increasing when I change a certain setting.

This solution also offers Mago a competitive edge in the sustainable waste management industry, because it allows them to scale their business without worrying about bottlenecks, and improves the service quality of their clients.

Requirements

ID	Category	Subcategory	Detail	MoSCoW
1	User Management	User Creation and Management	Intuitive forms for creating users in Admin Interface.	Must
2	User Management	User Creation and Management	Validate email addresses during user creation in Admin Interface.	Must
3	User Management	User Creation and Management	User-friendly interface for password and email change in User Interface.	Should
4	User Management	User Assignment and Roles	Assign devices to users in Admin Interface.	Must
5	Device Management	Device and Device Types Management	Forms for device creation.	Must
6	Device Management	Device and Device Types Management	Forms for device type creation.	Must
7	Device Management	Device and Device Types Management	Dropdowns or lists to change deviceType.	Must
8	Device Management	Active Device Management	Intuitive selector for active devices. (User)	Should
9	Device Management	Device Settings and History	Intuitive interface to display and modify active device settings, incorporating viewedBy & editedBy fields.	Should
10	Device Management	Admin Functionalities	Develop a comprehensive, user-friendly admin interface.	Must
11	Device Management	Settings Management	Interfaces for settings with adequate rights management.	Should
12	Real-Time Monitoring	Metrics Visualisation	Responsive and clear graphs for each metric.	Must
13	Real-Time Monitoring	Metrics Visualisation	Intuitive options for weekly, monthly views or zooming.	Should
14	Real-Time Monitoring	Metrics Management	Interface for displaying the latest value of active device's settings.	Should
15	Real-Time Monitoring	Metrics Export	Options for exporting data as CSV.	Could
16	Firmware Update	Update Management	Develop secure, user-friendly interfaces for file uploads to the server.	Could
17	Testing and Validation	Functionality Testing	Ensure that all features are functioning as intended and fix any bugs that may arise.	Must
18	Testing and Validation	User Experience Testing	Test the user interface for intuitiveness and user friendliness. Optimise the interface based on feedback.	Must
19	Documentation	Developers' Documentation	Maintain detailed, clear documentation for developers outlining system architecture, codebase, and functionalities.	Must

20	Documentation	Users' Documentation	Develop clear, user-friendly documentation and guides to assist users in navigating and utilising the application.	Should
21	Alert System	Notifications	Users should be notified depending on the status of their machines	Should
22	Alert System	Notification Management	Users should be able to select when they want to be notified	Should
23	Sustainability	Sustainability	Insights about the impact they have on the environment	Should

Testing methods

Linting

In the realm of white-box testing, meticulous attention to detail was dedicated to the application's source code through the implementation of linting, a fundamental facet of static analysis. Rather than relying on direct libraries, a strategic approach involved the utilisation of integrated development environments (IDEs) tailored to the distinct stacks employed throughout the project.

For the .NET portion, JetBrains Rider IDE emerged as the tool of choice, utilising its adept capabilities to uphold and enforce specific coding conventions inherent to the language. Similarly, for Golang, JetBrains Goland was used, ensuring a seamless and standardised coding experience. Simultaneously, PyCharm, another powerful JetBrains IDE, played a crucial role in maintaining the integrity of our Python codebase, ensuring consistency and maintainability.

By leveraging the robust features offered by Rider for .NET, Goland for Golang, and PyCharm for Python, the development team not only ensured compliance with language-specific coding conventions but also instilled a sense of cohesion and clarity in the code structure. This approach translates into heightened readability, reduced complexity, and enhanced collaboration among developers, ultimately contributing to the overall efficiency of the software development lifecycle.

Unit testing

The primary testing methodology applied to the source code involved the implementation of white-box testing, specifically unit testing. This approach facilitates the isolated examination of individual components within the application, ensuring their adherence to expected functionality. The objective with the unit tests was to examine each microservice and its components on an individual basis, ensuring their seamless operation in alignment with the intended functionality. The unit tests have been executed by the development team.

The examined elements of the application included testing models, repositories, services, and authentication procedures.. These are the most important components which make up the structure of the application as well as its functionality. Since the team worked with different kinds of stack (programming languages & frameworks), different libraries were applied for unit testing, however, the result was the same. These unit tests were properly applied based on predefined criteria, testing error handling scenarios and expected outcomes. Essentially, the tests were designed to address deviations from intended behaviour and validate the attainment of expected results.

Examples include ensuring that a logged-in user has smooth access to certain program portions while performing correct authorization checks. Furthermore, the testing framework included scenarios where users attempted to access unauthorised information or input invalid data for non-existent entities, increasing the robustness of the software.

The overarching objective was to attain a minimum of 60% unit test coverage. This means that around 60% of the application's components are expected to be thoroughly tested, aligning with the requirements stipulated by both university faculty and the client. However, expectations were surpassed, achieving a test coverage exceeding 70% across each microservice and its associated components.

Integration testing

The utilisation of integration testing, a black-box testing method was also applied in combination with unit testing. Unlike unit testing, the goal of integration testing was to test the whole software module, examining the functionality of two or more modules simultaneously. The ultimate goal was to focus on validating the interaction between components, making sure that it has a robust performance of the entire system. The integration tests have been executed by the development team.

In the context of integration testing, the focus was to primarily test the orchestrators. These services, directly interacting with all microservices, were assessed to guarantee a smooth flow of data between them, effectively merging their functionalities. The goal was to guarantee that orchestrators were smoothly integrated to enable efficient data transmission across microservices and effective operation of related modules.

A notable example employed in the application involves an orchestrator using the metrics microservice and device microservice. The orchestrator orchestrates the retrieval of information from both microservices, even though they operate independently without knowledge of each other. This approach ensures that the orchestrator gathers information from both microservices, ultimately combining them to return a comprehensive and well-structured data for the API user to access. On top of it, the orchestrators also ensure that various errors that may occur in those microservices get handled accordingly.

Understanding the full extent of coverage for integration tests can be complicated in complex systems where multiple components interact with each other. Our primary goal with these tests was not to achieve 100% coverage across the entire infrastructure, as this can be expensive and time-consuming. Instead, we focused on ensuring robustness between all components. It's important to note that while this doesn't guarantee complete resilience of the system, it significantly strengthens our confidence in the functionality and integrity of the orchestrators. Each aspect of the orchestrators was tested with a variety of inputs, ensuring their performance under different scenarios, which is a vital part of validating their overall reliability and effectiveness.

End 2 end testing

Similar to its role in integration testing, the SpecFlow framework adeptly served as a tool for behaviour-driven development while effectively simulating end-to-end tests. Within the SpecFlow paradigm, test scenarios were meticulously crafted using Gherkin syntax, providing a natural language format for articulating test cases in plain English. These scenarios seamlessly translated into step definitions, coded in .NET C#, where the intricate test logic found its implementation.

The scope of these tests extended beyond mere validation of logic within the orchestrators; rather, they meticulously verified the orchestration logic alongside the associated microservices. This comprehensive approach ensured not only the robustness of the orchestrator but also the seamless coordination and interaction with its microservices and all of the API endpoints that are connected to it.

Moreover, the simulation encompassed an array of database transactions, scrutinising the concurrent operations throughout the entire process. In essence, each facet of logic and transactional flow underwent meticulous testing, rendering this methodology a highly effective means for conducting end-to-end tests.

Risks and mitigations

Security

Cloud security risks encompass potential vulnerabilities and threats that may compromise the confidentiality, integrity, and availability of data and services stored or processed in the cloud. These risks include unauthorised access, data breaches, insecure APIs, and more. Mitigating these risks is crucial to ensure the overall security of cloud-based systems.

To counteract these risks, Auth0 has been employed. Auth0 enables organisations to implement access control policies. This ensures that users have the appropriate level of access to resources, reducing the risk of unauthorised data access or modification. Furthermore, Auth0 utilises secure tokenization and encryption techniques to protect sensitive user data and credentials during transmission and storage. This helps prevent data breaches and ensures the confidentiality of information stored in the cloud. Auth0 is designed to be scalable and reliable, ensuring that authentication and authorization processes remain effective even as cloud-based systems expand. This helps maintain security measures in dynamic and growing environments.

Outages

Cloud outages are instances when cloud computing services become temporarily unavailable, disrupting access to applications, data, and other resources hosted on the cloud. These outages can result from various factors, including hardware failures, software glitches, network issues, or even natural disasters. Mitigating the impact of cloud outages is essential to ensure the reliability and availability of services.

By employing health and ready checks, administrators will be notified when issues are detected, enabling them to proactively address potential problems before they escalate. A health check will check whether a database or service is approachable. A ready check will check whether the database or service is ready to handle transactions.

Performance

Performance issues are a significant concern, coming from slow response times, degraded system responsiveness, and inefficient resource utilisation. These issues can greatly impact the reliability and scalability of cloud services, leading to a negative user satisfaction and trust. A critical aspect of maintaining optimal performance in such environments is the effective management of resources, particularly ensuring that the system's memory and processing power are being utilised efficiently.

To tackle these challenges, horizontal scaling is employed as a strategic solution. This approach involves augmenting the system with additional nodes, or 'pods', to distribute the workload more evenly, thereby enhancing overall system performance and responsiveness. The process is triggered when the system's CPU usage reaches a predefined threshold, set at around 80%. This scaling not only helps in managing the load but also plays a vital role in preventing any single node from becoming a bottleneck. Additionally, the memory usage of each service is monitored to ensure that it is allocated appropriately and efficiently. This focus on CPU and memory utilisation is crucial for maintaining a balanced and high-performing system.

Alongside managing CPU resources, monitoring the response time of each service is also important. The goal is to keep these response times within a certain range, less than 500 milliseconds. By actively monitoring and analysing these performance metrics, it becomes possible to identify and address any issues, thereby preventing them from escalating into more significant problems. This proactive approach to monitoring not only encompasses response times but also memory usage, ensuring that each service has access to the resources it needs.

Stability of the production environment during testing

Testing in the production environment can lead to unexpected outages or disruptions that directly affect end-users. Users may experience service interruptions, slow performance, or errors during the testing. Furthermore, testing in a live production environment poses the risk of unintentional data modifications or loss. It may also expose sensitive information to unintended parties.

This risk has been mitigated by setting up a test environment. A dedicated test environment provides a controlled and isolated space for testing changes and updates. Testing in an environment that mirrors production conditions allows for performance testing to evaluate how the system behaves under various loads. In the case unexpected issues arise during testing, it is easier to roll back changes.

Automation and deployment

GitHub Actions

To deploy the source code onto the OpenShift environment, GitHub Actions, a CI/CD platform, has been used. GitHub Actions offers automation for application builds, testing, and deployment through automated pipelines. Through this tool, workflows could be crafted, which provide a configurable automated process that will run multiple tasks in a sequential manner.

Specific workflows were established for each microservice and orchestrator in the application. This approach empowered us to execute essential steps like building, testing, creating a containerized image of the application, and deploying the image to Open-Shift, seamlessly with each source code modification.

In our development process, we use two different branches within our repository, each serving a specific purpose. The 'development' branch is where changes are initially made. Once a developer commits changes to this branch, an automated deployment process is triggered, deploying these changes to the testing environment (first runs unit tests, pushes to docker registry, and then deploys a new rollout). This setup allows developers to thoroughly test the newly deployed features in a controlled setting, ensuring that everything operates as intended. After satisfactory testing in this environment, the next step involves the developer creating a pull request to merge the changes into the 'main' branch.

This pull request to the main branch is an important and required step as it activates the integration tests. These tests are designed to evaluate the system in the test environment, ensuring that all components work together and that the new changes do not affect existing functionalities in a negative way. Only after these integration tests pass, indicating that everything in the test environment is functioning correctly, are changes permitted to be merged into the production environment.

After merging the changes into the main branch, a new rollout process begins, pushing these updates to the production environment. To enhance system reliability, we've implemented a rollback feature within our deployment pipeline. This feature actively monitors the new rollout for a period of approximately two and a half minutes, checking for any errors or issues. If any problems are detected during this window, or if the rollout is not successful, the rollback feature is automatically triggered. This mechanism reverts the system to the previous, stable version of the code, thereby minimising disruptions and maintaining the integrity of the production environment. This combination of a structured branching strategy, rigorous testing, and the safety net provided by the rollback feature forms the backbone of our approach to reliable and efficient software deployment.

Bicep templates & deployment scripts

Azure services have been used to deploy databases and an Azure Function in the Azure cloud environment. To leverage the efficiency of automation, a Bicep template was employed, which is a domain-specific language designed explicitly for deploying Azure resources onto the Azure platform. This Bicep template serves as a robust Infrastructure as Code (IaC) solution within the Azure ecosystem, providing a structured and reproducible method for orchestrating the deployment of resources.

For the Bicep template, a series of resources have been constructed, which are required to set-up all the databases and the Azure Function along with their respective configuration onto the resource group. Furthermore, a deployment script was utilised which ensured that all of the secrets and passwords were given to the Bicep along with the source code that should be deployed onto the resources within Azure.

Environmental isolation

A multi-environmental approach has been chosen to address various problems of the project's life cycle. The primary goal behind this decision was to mitigate risks. By separating testing, and production environments, the team established a systematic framework for identifying and rectifying issues during the testing phase, ensuring the stability of the production environment.

Moreover, the utilisation of separate environments serves the purpose of isolating changes during the development process. This approach allows for experimentation and iteration without impacting the live production environment, ultimately providing a focused and controlled development process. The clear separation of concerns allows for efficient collaboration among development, testing and operational teams, which would be extremely beneficial if Mago was to grow as a company and expand their branch.

Another major benefit of this approach is the fact that it ensures scalability and performance of the system. Through a simulation of real-world scenarios, the developer is able to assess the system's ability to handle increased loads and optimise its performance accordingly. This approach does not only reduce downtimes, but also saves companies from a loss of revenue.

Observability

The cloud application's uptime is monitored to ensure a seamless user experience, with a particular focus on several key metrics: the number of HTTP requests, HTTP response time, the total number of threads, total known allocated memory, and the health status of the system. This monitoring includes tracking the frequency and efficiency of HTTP requests, which is crucial for assessing server load and user traffic. The response time of these requests is also measured, providing vital information about the application's speed and responsiveness, which directly impacts user satisfaction. Additionally, the system keeps a close eye on the total number of threads, an important factor in understanding the application's capacity for concurrent processing. Monitoring the total known allocated memory is essential to ensure the application operates efficiently and without memory-related bottlenecks. The health status provides a snapshot of the application's overall operational health, while the readiness status is specifically monitored to determine the system's preparedness to handle requests, an essential factor in maintaining continuous service availability. By implementing the Prometheus monitoring system and integrating it with Grafana, these metrics are visualised through charts and graphs, offering clear insights into the application's performance and stability.

Conclusion and Advice

In conclusion, this project has not only been an educational experience but also a practical contribution to the area of sustainable waste management through cloud computing. Our development of the eComposter Cloud Solution shows that with the right combination of technology, strategy, and teamwork, great achievements can be made in operational efficiency and service delivery. The implementation of this cloud-based solution, made specifically for Mago Bio Solutions, has streamlined the management of eComposter machines, reducing manual interventions, and enabling scalable growth.

However, the development of such a solution is a never ending process. As technology evolves and client needs change, the solution must adapt accordingly. Continuous monitoring, testing, and updating are crucial for maintaining the security, performance, and user-friendliness of the system.

Therefore, our advice for future development includes:

- **Stay Agile and Adaptable:** The tech world is constantly evolving. Keep the system architecture and design flexible to integrate new technologies and meet changing client demands.
- **Focus on Security:** As more data gets processed and stored in the cloud, prioritise security measures to protect against breaches and maintain user trust.
- **Emphasise User Experience:** Regularly gather user feedback and make iterative improvements to enhance usability and functionality.
- **Invest in Continuous Testing and Monitoring:** Implement robust testing and monitoring strategies to ensure system stability, performance, and quick resolution of any issues.
- **Plan for Scalability:** Design the system to handle growth, both in terms of the number of users and the volume of data processed.

This project has been an invaluable learning experience for all of us, offering practical insights into the application of cloud computing in real-world scenarios. We believe that the eComposter Cloud Solution sets a new standard in the sustainable waste management industry and serves as a solid foundation for future innovations in this field.