Guilherme Pinto Luz
Marmeleiro Malhado

APPJET PLATFORM

Open-source platform for deployment automation and configuration over web infrastructures.

Project dissertation as a requirement for obtaining the degree of **Master of Software Engineering**.

Advisor
Professor Doctor João Ventura

Jury Panel
President (Professor Doctor Cláudio Sapateiro, Politécnico de Setúbal)
Examiner (Professor Doctor Valentim Realinho, Politécnico de Portalegre)

October, 2024

I dedicate this project to my family, especially highlighting my parents and grandparents, sister, girlfriend and her parents and my closest friends, endless sources of support and inspiration.

Acknowledgements

I would like to express my sincere gratitude to some individuals and institutions whose support was crucial to the completion of this work:

To my advisor, Professor Doctor João Ventura, for his invaluable guidance, unwavering encouragement and for generously sharing knowledge beyond what was required for the development and completion of this project.

To my family, for consistently supporting me and motivated me during and before the master's degree, for the moments of relaxation in-between the study time, that helped alleviate the pressure that I had during the last two years.

To my girlfriend and her parents, for being always there for me, for the unconditional support, motivational attitude, and friendship.

To the School of Technology of Setúbal and to the Polytechnic Institute of Setúbal, for providing the resources necessary for conducting this research and development of the project.

To Uniksystem, on the person of João Costa, for pushing and allowing me to start the master's degree while I was Java Tech Lead, contributing to a healthy work environment with full Support specially.

To Kuehne+Nagel, my current company, for being there always motivating me to conclude the master's degree, for motivating me each day to become better and better.

To my close friends, for their mutual support and relaxing moments during this last two years.

At last but not least, to all the professors who offered their valuable suggestions and insights.

Thank you.

Resumo

Appjet é um projeto destinado a simplificar a configuração de servidores e os processos de implementação de deployments de aplicações web. Aborda os desafios enfrentados em configurações manuais e cadeias de ferramentas complexas, fornecendo uma plataforma amigável ao utilizador.

O objetivo principal do projeto é automatizar estes processos, reduzindo o tempo e o esforço para os programadores, ao mesmo tempo que melhora a qualidade da aplicação. Ao formalizar os detalhes da aplicação em ficheiros de configuração, Appjet simplifica os fluxos de trabalho de implementação tanto para indivíduos como para equipas. Como uma iniciativa de código aberto, incentiva contribuições da comunidade e potenciais modelos de negócio relativos ao *hosting* de instâncias da plataforma.

O desenvolvimento do projeto seguiu uma metodologia Kanban, dando um elevado enfâse à visualização de tarefas, o trabalho limitado em progresso e a melhoria contínua. Além disso, integra tecnologias modernas para garantir eficiência e confiabilidade, com testes no mundo real e documentação abrangente. As funcionalidades incluem uma aplicação cliente para configuração e implementação, um frontend de monitorização de servidores, mecanismos de autenticação de clientes e um site para documentação e tutoriais. Em última análise, Appjet visa facilitar processos de deployment de aplicações web mais rápidos e simplificados.

Palavras-chave: Deployments, Configuração de Servidores, Automação, Código-Aberto

Abstract

Appjet is a project aimed at simplifying server configuration and web application deployment processes. It addresses the challenges faced in manual setups and complex toolchains by providing a user-friendly platform.

The project's primary goal is to automate these processes, reducing time and effort for developers while improving application quality. By formalizing application details into configuration files, Appjet streamlines deployment workflows for both individuals and teams. As an open-source initiative, it encourages community contributions and potential business models around hosting instances of the platform.

The project follows a Kanban methodology, emphasizing task visualization, limited work in progress, and continuous improvement. Additionally, it integrates modern technologies to ensure efficiency and reliability, with real-world testing and comprehensive documentation. Features include a client application for configuration and deployment, server monitoring frontend, client authentication mechanisms, and a website for documentation and tutorials. Ultimately, Appjet aims to facilitate faster, more streamlined deployment processes for web applications.

Keywords: Deployments, Server configuration, Automation, Open-Source

Index

List of Figures

List of Tables

List of Acronyms

- ABAC - Attribute-Based Access Control
- ACLI - Appjet Command Line Interface
- API - Application Programming Interface
- APP - Application
- AWS - Amazon Web Services
- CD - Continuous Delivery
- CI - Continuous Integration
- CLI - Command Line Interface
- CPU - Central Processing Unit
- CSRF - Cross-Site Request Forgery
- DEVOPS - Development and Operations
- DOM - Document Object Model
- HTTP - Hypertext Transfer Protocol
- HTTPS - Hypertext Transfer Protocol Secure
- I/O - Input/Output
- JSON - JavaScript Object Notation
- JS - JavaScript
- JWT - JSON Web Token
- MFA - Multi-Factor Authentication
- ML - Machine Learning
- OS - Operative System
- OSS - Open Source Software
- PAAS - Platform as a Service
- QUIC - Quick UDP Internet Connections
- RAM - Random Access Memory
- RBAC - Role-Based Access Control
- REST - Representational State Transfer
- SCP - Secure Copy Protocol
- SPA - Single Page Application
- SSH - Secure Shell
- SSL - Secure Sockets Layer
- SVN - Subversion
- TLS - Transport Layer Security
- UI - User Interface
- URL - Uniform Resource Locator
- UX - User Experience
- VCS - Version Control System
- WWW - World Wide Web
- XML - Extensible Markup Language
- XSS - Cross-Site Scripting

Chapter 1. Introduction

This chapter examines the contemporary landscape of server configuration and application deployment, focusing on the technologies, methodologies, and practices that define the current state of the field. Our objective is to understand the challenges posed by manual and complex tool-based approaches, as well as to identify potential areas for innovation and improvement. By analyzing existing solutions and their limitations, we aim to highlight the opportunities that platforms like Appjet seek to address.

The project, **Appjet**, aims to develop a platform that simplifies server configuration and streamlines the deployment of web applications. This initiative arises from the widespread challenges associated with these processes, which are critical stages in software development and maintaining online software availability. The growing complexity of manual server setups and application deployments makes these tasks not only time-consuming but also error-prone, which can hinder productivity and delay releases [45].

While larger-scale applications may rely on sophisticated, complex toolchains to automate deployments, such solutions are often excessive and impractical for smaller projects or individual developers. This creates a significant gap between manual processes and the advanced systems that require substantial resources and expertise. **Appjet** seeks to bridge this gap by offering a middle-ground solution that is accessible, yet effective. In this chapter, we will introduce the problem in more detail, outline the project's goals and expected outcomes, and provide a brief overview of the chosen methodology.

Server configuration and web application deployment are essential but often labor-intensive aspects of the software development lifecycle. For developers, especially those working on small to medium-scale projects, manual implementations are not only time-consuming but also susceptible to errors, leading to longer deployment cycles and potential downtime [46]. On the other hand, more complex toolchains designed for large-scale operations are overkill for these developers, requiring advanced knowledge and infrastructure.

Appjet addresses these challenges by offering a streamlined, automated solution that simplifies deployment workflows. Whether the user is an individual developer or part of a small team, the platform aims to reduce the friction caused by manual processes while avoiding the steep learning curve and heavy resources demanded by larger-scale tools.

The primary goal of **Appjet** is to automate tedious manual tasks and simplify the overall deployment process, allowing developers to focus more on writing code and less on infrastructure management. By formalizing application configurations through declarative files, Appjet speeds up deployment cycles, reduces the risk of errors, and enhances the usability and quality of web application deployments.

Additionally, as an open-source platform, **Appjet** fosters community contributions, allowing users to continuously refine the tool and develop new features. This model also presents the opportunity for sustainable business ventures, such as hosting Appjet instances as a service for users who prefer not to manage their own deployments.

The expected results of this project are clear:

A significant reduction in the time and effort required to configure and deploy web applications.

- Streamlined deployment workflows, leading to faster and more efficient deployment cycles.
- Enhanced quality of deployments through process automation, reducing errors and ensuring consistency.
- Increased community engagement via contributions, which will drive ongoing improvements to the platform.
- The potential establishment of sustainable business models, particularly through hosted versions of Appjet that could be offered as a service.

The project will adopt an Agile methodology to ensure flexibility and continuous improvement during development. Within this framework, **Kanban** will be used to manage tasks and track progress visually, from setting up the GitHub repository to developing Appjet's core services and client application. This approach emphasizes efficiency and transparency, allowing for easy monitoring of ongoing work and facilitating quick adjustments based on feedback.

Continuous integration of modern technologies will ensure that new features are tested and refined in real-world conditions. Documentation will be created alongside development, including resources for server monitoring, client authentication, and tutorials to guide users through the platform. The Agile process will allow iterative cycles of improvement, driven by both internal testing and community contributions.

Additionally, the project will incorporate real-world testing and user feedback as a key part of the development cycle. This will involve deploying early versions of Appjet in various environments to identify potential issues and optimize performance under different conditions. Regular **user testing** will help ensure that the platform is intuitive and aligns with the needs of developers. By actively engaging the community, Appjet will gather insights that will shape future releases, ensuring that new features and improvements are relevant and impactful. This user-centric approach, combined with Agile's iterative cycles, will foster a dynamic development process that remains responsive to evolving project goals and user expectations.

Additionally, Appjet will incorporate modern development practices such as continuous integration (CI) and continuous deployment (CD). These practices will ensure that new features are rigorously tested and optimized before release. By fostering a culture of continuous improvement, the project will remain responsive to evolving industry standards and user expectations.

IEEE studies reinforce the importance of simplifying deployment processes to enhance developer productivity. For instance, a 2021 IEEE report on cloud-native applications highlights how declarative configurations significantly reduce deployment times by enabling automated environment provisioning [IEEE 2021]. Another study emphasized the role of open-source platforms in democratizing advanced deployment capabilities, making them accessible to smaller teams with limited resources [IEEE 2020]. These findings underscore the relevance of Appjet's goals and approach in addressing contemporary challenges in application deployment.

By integrating insights from IEEE articles, Appjet aligns its development strategy with proven methodologies and industry best practices. This approach not only validates the platform's objectives but also strengthens its potential impact on the software development community.

Chapter 2. State of the Art

In this chapter, we delve into the contemporary landscape of server configuration and application deployment, exploring the existing technologies, methodologies, and practices that shape the field. Our focus is on understanding the current state of the art in addressing the challenges associated with manual and complex tool-based approaches, identifying opportunities for innovation and improvement.

Overview of the Research Topic

The research topic centers on the optimization of server configuration processes and the acceleration of web application deployment. These processes, critical in software development and delivery, often present significant bottlenecks and complexities that hinder efficiency and productivity. Our aim is to explore the latest advancements and trends in this domain, seeking insights and inspirations to inform the development of our platform, Appjet.

Purpose and Scope of the Dissertation

The purpose of this dissertation is twofold: first, to provide a comprehensive review and analysis of the state-of-the-art technologies and methodologies relevant to server configuration and application deployment; and second, to evaluate the potential of our proposed solution, Appjet, in addressing the identified challenges and advancing the field. Through this exploration, we aim to contribute to the ongoing dialogue surrounding best practices and innovations in server management and application deployment.

Research Objectives

In order to achieve our goals, we have set forth the following research objectives such as to examine the current landscape of server configuration and application deployment technologies, methodologies and related tools.

To identify key challenges and limitations associated with existing approaches, including manual configurations and complex toolchains and also evaluate recent advancements and emerging trends in the field, such as containerization, automation, and DevOps practices.

To assess the suitability and potential impact of our proposed solution, Appjet, in simplifying server configuration and streamlining application deployment processes.

To explore potential use cases, deployment scenarios, and implications of Appjet in real-world settings, including its applicability in individual and small team environments.

By addressing these objectives, this chapter, named as State-of-the-Art aims to provide a comprehensive overview of the state of the art in server configuration and application deployment, laying the groundwork for the subsequent exploration and analysis of our proposed solution, Appjet. Through this inquiry, we endeavor to contribute valuable insights and perspectives to the broader discourse on software development practices and technologies.

Foundational Technologies

Golang

In recent years, the Go programming language, commonly referred to as Golang, has emerged as a formidable force in the realm of software development. Developed by Google in 2007 and released to the public in 2009, Golang was conceived with the aim of addressing the challenges faced by large-scale distributed systems, such as those encountered in Google's own infrastructure. Since its inception, Golang has garnered widespread acclaim for its simplicity, efficiency, and concurrency support, making it a popular choice among developers for a diverse range of applications. Go possesses an official website hosted on https://go.dev/ [1]-[3].

Overview of Golang:

At its core, Golang is a compiled typed, compiled programming language that aims for simplicity, performance and robustness. Drawing inspiration from established languages such as C and Python, Golang combines the low-level control and efficiency of C with the high-level productivity and readability of Python. The result is a language that strikes a delicate balance between power and ease of use, making it accessible to both non-experienced and experienced developers alike.

Key Features and Advancements

Golang's success can be attributed to several key features and advancements that set it apart from other programming languages:

- **Concurrency Support:** Golang's lightweight goroutines and channels provide first-class support for concurrent programming, allowing developers to write highly scalable and efficient concurrent systems with ease [4].
- **Performance Optimization:** Golang's compiler and runtime optimizations contribute to its impressive performance characteristics, a good option for high-performance applications [5].
- **Standard Library:** Golang's extensive standard library provides a rich set of built-in packages and modules for common tasks and protocols [6].

Applications and Use Cases

Golang's versatility and efficiency have led to its adoption in a variety of application domains, including:

- Web Development: Golang's fast compilation times and efficient runtime make it well-suited for building high-performance web applications and APIs [7].
- Systems Programming: Golang's low-level control and minimal runtime overhead make it ideal for systems programming tasks [8].
- Cloud Computing: Golang's concurrency support and efficient resource utilization make it a natural fit for cloud-native development [9].

In summary, Golang continues to evolve as a versatile and efficient programming language, with widespread adoption across various industries and domains. Its simplicity, efficiency, and concurrency support make it suit well for a wide range of applications, from web development to systems programming to cloud-native development. As Golang continues to mature and gain attraction in the software development community, it is poised to play an increasingly important role in transforming, improving and shaping the future of software development.

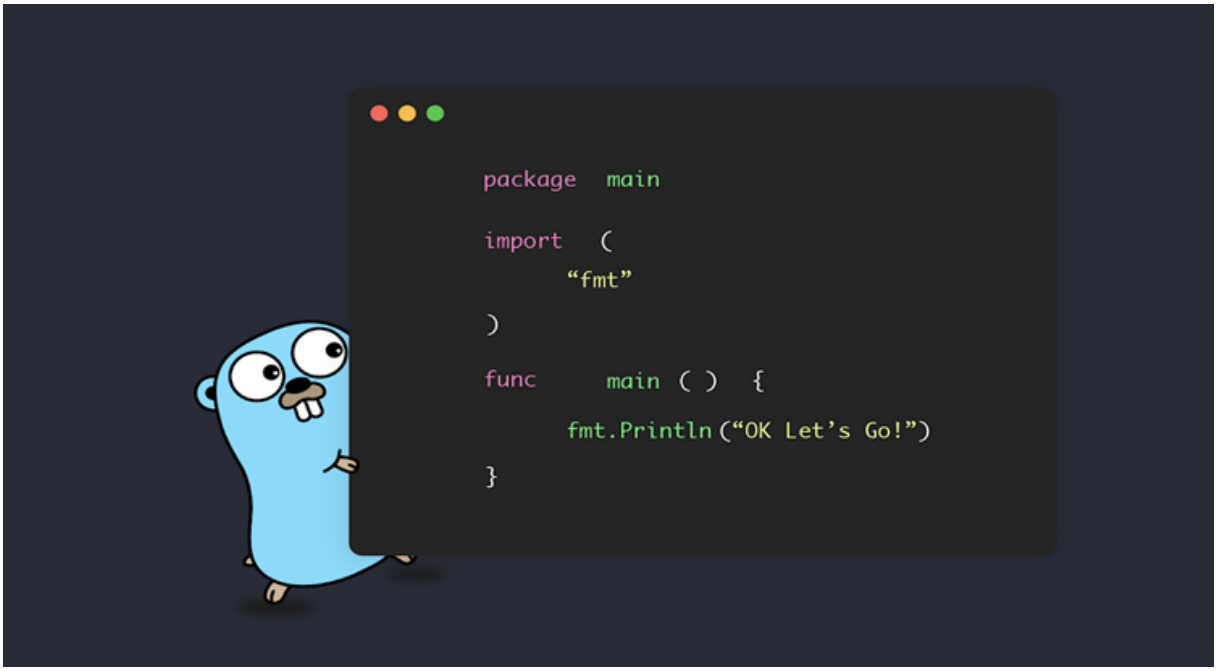

Figure 1. Example of Golang 'hello world' [47]

This image simply reflects the language simplicity, “c like style”. Which enabled us to easily understand the language and develop the full project in both high and low level programming.

Understanding the HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is part of the foundation of communication on the World Wide Web (WWW), providing the inter client/server exchange of information. Over the years, HTTP has undergone significant advancements to enhance its performance, security, and flexibility, reflecting the evolving needs of internet-based applications.

HTTP/1.1, prevalent for over a decade, introduced features like persistent connections and chunked encoding to improve data transfer efficiency. However, with the increasing complexity of web applications, it exhibited limitations in scalability and performance.

HTTP/2, introduced in 2015, addressed these shortcomings by enabling concurrent streams within a single connection through techniques like multiplexing and header compression. Server push functionality further enhanced performance.

HTTP/3, ratified in 2020, built upon the QUIC protocol, promises even faster and more reliable communication by minimizing latency and improving congestion control, particularly in scenarios with packet loss or network congestion.

Security remains paramount in HTTP protocol development, with HTTPS as the standard for secure web communication. Efforts include deprecating vulnerable SSL/TLS versions and promoting stronger cryptographic algorithms.

The proliferation of web APIs and microservices architectures led to widespread adoption of HTTP for building RESTful services. Adherence to REST principles ensures API consistency, scalability, and ease of use.

HTTP status codes and verbs play a crucial role in communication between clients and servers. Status codes indicate the outcome of HTTP requests, while verbs define the actions performed by clients. Common status codes include 200, 404 and 500 while verbs/request methods include GET, POST, PUT, and DELETE. As shown on the picture below.

Despite gradual adoption challenges, HTTP/3 implementation is underway across major web servers, browsers, and content delivery networks, promising broader adoption in the near future. In conclusion, the evolution of HTTP underscores a commitment to improving performance, security, and interoperability in internet-based communication. Staying informed about emerging standards and best practices is essential on developments of new applications for harnessing the full potential of HTTP [10-17].

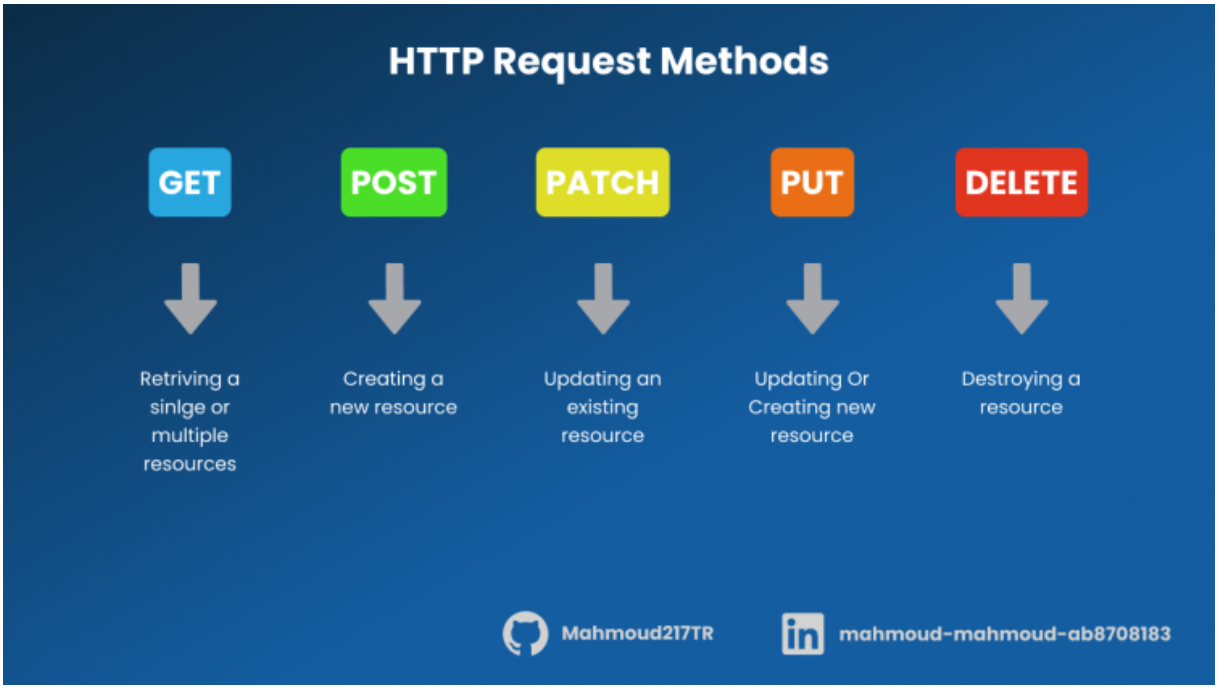

Figure 2. HTTP Request Methods [48]

These request methods, as described on the figure above, combined with usefull status codes, should be essential in any API as they exist in every web engineer book, and documentation, as Hello World of Web Development. A Software that does not represent this, tends to be harder to maintain and also could lead to unexpected bugs.

REST Principles

REST, or Representational State Transfer, is a common architecture style in distributed systems. REST architectures or RESTful systems are characterized by a set of rules, including the absence of state (stateless) and the existence of client-server communication, as well as an independence between the client and server. This means that changes to the server will not affect the client and vice versa, as long as both sides understand the communication protocol and message format they exchange [18].

In this architecture, also described on the figure below, the client sends http requests, with data representation in JSON or XML, to receive or modify data, and the server (or servers) send responses to these requests. For a request to exist, it must have an HTTP method (HyperText Transfer Protocol), which defines the requested operation, a header that describes the content to be requested or required, the path of the resource, and finally a body that may or may not contain additional data about something to be sent or received [19].

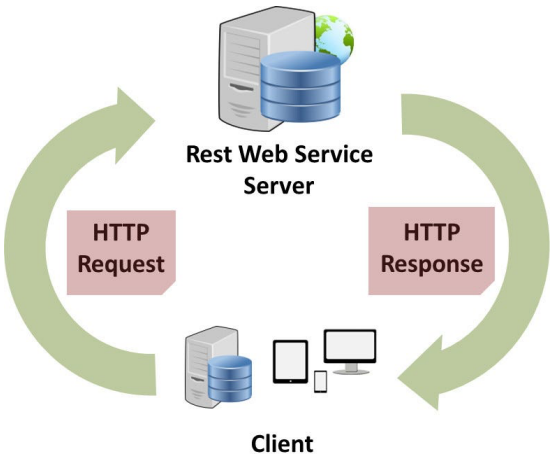

Figure 3. Client/Server Architecture using a Rest API

Basics of SCP (Secure Copy Protocol)

SCP or Secure Copy Protocol, is a network protocol used for security matters to transferring files between a local and a remote host. It operates over SSH (Secure Shell) and provides encryption and authentication to ensure data confidentiality and integrity during transmission.

- **Encryption**: SCP encrypts data during transmission, preventing unauthorized access or tampering. It utilizes cryptographic algorithms to ensure the confidentiality of transferred files.
- **Authentication**: SCP authenticates users using SSH keys or passwords, ensuring that only authorized individuals can access and transfer files.
- **Integrity**: SCP verifies the integrity of transferred files by performing checksums, ensuring that files remain unchanged during transmission.
- **Usage**: SCP is commonly used in Unix-like operating systems for secure file transfer. It provides a command-line interface for transferring files between hosts, offering simplicity and ease of use [20-22].

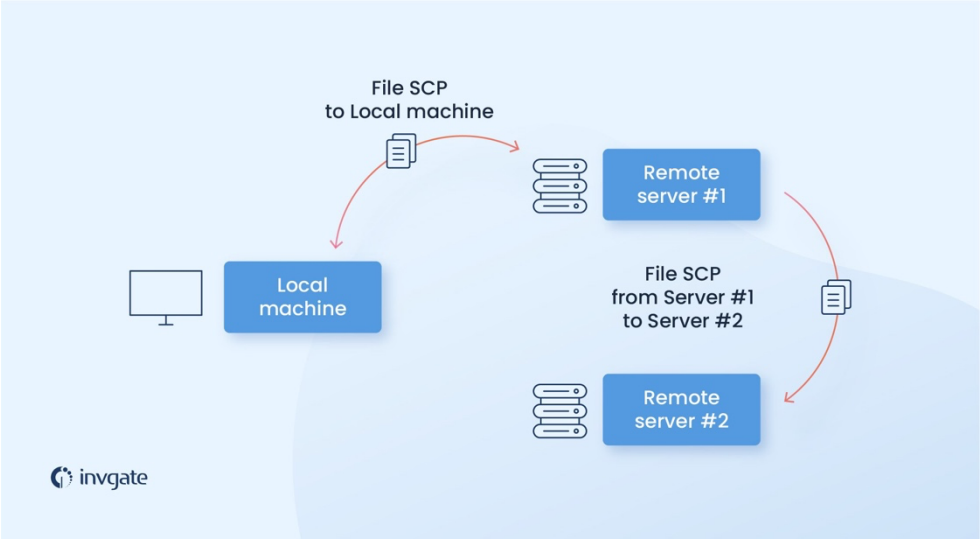

Figure 4. SCP workflow and behaviour diagram [49]

Version Control with Git

Version control with Git is a widely used system for tracking changes in files and coordinating work among multiple collaborators in software development projects.

Git tool is a distributed version control tool that enables developers or devops professional to manage changes to their codebase efficiently. It provides the following key features [23].

- **Distributed Architecture**: Differently than other centralized version control systems such as SVN, Git operates on a distributed model where each developer has a complete versioned copy of the repository source code, including its full history. This allows for offline work and decentralized collaboration.
- **Branching and Merging**: Git allows users to create lightweight branches to work on new features or fixes independently. Branches can be merged into the main branch, which usually is mater, once the changes are complete, facilitating parallel development and experimentation.
- **Commit History**: Git maintains a detailed history of commits, including who made the changes, when they were made, and why they were made. This provides transparency and accountability, making it easier to track down issues and understand the evolution of the codebase.
- **Staging Area**: Git introduces a staging area where changes can be selectively added before committing them to the repository. This allows for more granular control over which changes are included in each commit.
- **Collaboration**: Git facilitates collaboration among team members by providing mechanisms for sharing changes between repositories, such as push, pull, and fetch operations. It also supports workflows like Pull or Merge Requests (depending on gitlab or github) for reviewing and discussing changes before merging them into the master branch [24-26].

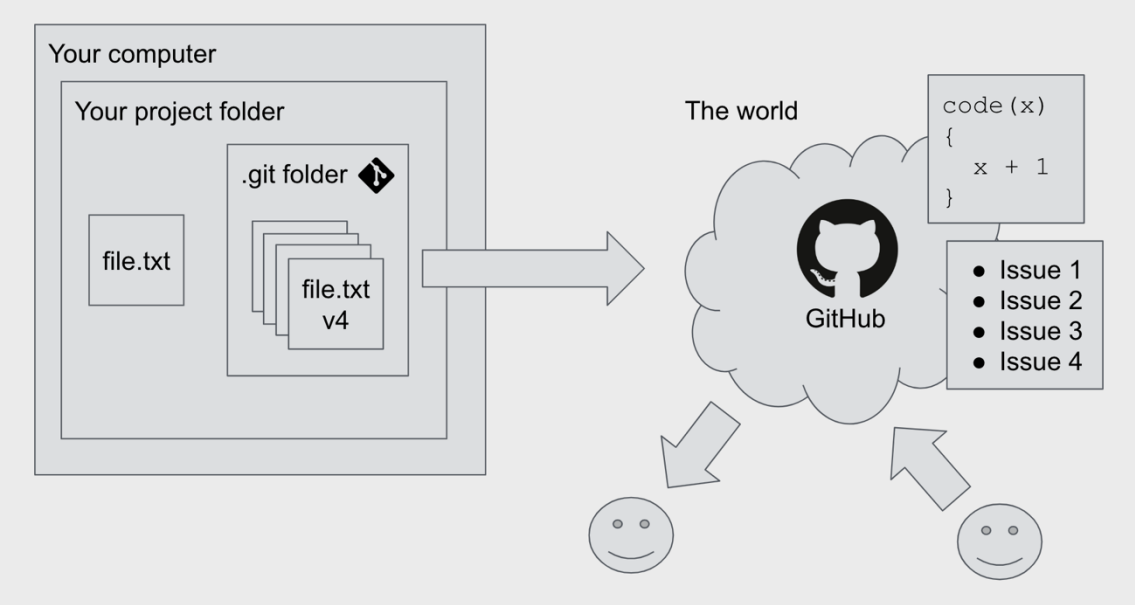

Figure 5. Git Repository usage in a real work experience scenario

In the previous image, the project, which is running on our local machines, is hosted remotely in GitHub, which also allow us to have an Agile Board, where we can view all the issues open and also the backlog available and ready for development. Almost all companies use a Version Control System, SVN could be a solution as well.

Data Handling and Formats

WORKING WITH JSON DATA

Working with JSON data has become a fundamental aspect of modern software development, especially in web and API development.

JSON, which stands for JavaScript Object Notation, is a lightweight data stucture format that is easy to read and write and easy to be consumed by machine parsers. It has become the standard for data interchange in web APIs, configuration files, and various other contexts. Key features of working with JSON data include:

- Data Representation: JSON allows for the representation of structured data using key-value pairs, arrays, and nested objects. This flexibility makes it suitable for encoding a wide variety of data types, including strings, numbers, booleans, arrays, objects, and null values.

- Serialization and Deserialization: Serialization is the process of converting data structures into a JSON string, while deserialization is the process of parsing a JSON string and converting it back into data structures in the programming language of choice. Most programming languages provide built-in or third-party libraries for serializing and deserializing JSON data.

- Interoperability: JSON facilitates interoperability between different systems and programming languages, as it is supported by virtually all modern programming languages and platforms. This makes it an ideal choice for data exchange in distributed systems and microservices architectures.

- Schema Flexibility: Unlike some other data formats like XML, JSON does not enforce a rigid schema. This flexibility allows for evolving data structures over time without breaking compatibility with existing consumers.

- Human Readability: JSON is designed to be human-readable and easy to understand, which makes debugging and troubleshooting data-related issues more straightforward [27-28].

```
{
 "FirstName"            : "Sam",
 "LastName"             : “Jackson”,
 "employeeID"           : 5698523,
 "Designation"          : "Manager",
 “LanguageExpertise”    : [“Java”, “C#”, “Python”]
 “Car”                  : “ “
}
```

Figure 6. Example of JSON structured file

Web Development Frameworks and Libraries

GoLang HTTP Library

The robustness and reliability of web applications are critical factors for their success. The Go programming language provides the net/http library as one of its core components, offering a robust, user-friendly, and efficient way to handle HTTP requests and responses. This library is widely adopted for building web servers, clients, and various network-based services.

It is a core component for building reliable web applications, offering a simple and efficient way to handle HTTP requests and responses. It includes a built-in HTTP server supporting HTTP/1.1 and HTTP/2, with concurrency managed by Go's goroutines.

Key features include middleware support for tasks like logging and authentication, extensibility through interfaces such as http.Handler, and advanced routing via community frameworks like Gorilla Mux and Chi. The library also natively supports HTTPS and TLS for security, and provides robust testing tools through the httptest package, along with profiling and tracing capabilities to optimize performance.

Implementing Authentication Technologies in Web Applications

Implementing authentication technologies in web applications is crucial for ensuring secure access to sensitive resources and protecting user data. Some of it's key aspects include:

- **User Authentication**: User authentication verifies the identity of users attempting to access a web application. Common authentication mechanisms include username/password authentication, social login (OAuth), and multi-factor authentication (MFA), which provide varying levels of security and user convenience.
- **Session Management**: Once users are authenticated, web applications typically create and manage user sessions to maintain their authenticated state. Session management involves techniques such as using cookies or tokens to track user sessions and enforce access controls throughout the user's interaction with the application.
- **Authorization**: The authorization determines what actions and resources authenticated users are allowed to access within the application and also the inverse, which resources are not allowed to access. Role-based access control, which stands for RBAC, and attribute-based access control, which stands for ABAC, are common authorization models used to define and enforce systems access policies based on user roles and profiles or other business specific attributes.
- **Security Best Practices**: Implementing secure authentication requires adherence to security best practices, such as using strong cryptographic algorithms, protecting against common vulnerabilities like cross-site scripting, which stands for XSS, and cross-site request forgery, which stands for CSRF, and securely storing and transmitting sensitive user credentials [31-32].

Containerization and Deployment

DOCKER FOR CONTAINERIZATION

Docker is a platform that allows developers to develop, deploy, and run applications using containers. Containers are lightweight, portable, and self-sufficient environments that encapsulate application code, runtime, libraries, and dependencies, as shown on figure 8.

Docker simplifies the process of building, deploying, and running applications by providing a consistent environment across different infrastructure and environments [33-35].

Implementing Docker in Software Development:

- **Containerization**: Docker enables containerization, which involves packaging applications and their dependencies into containers. Containers provide isolation, portability, and scalability, allowing developers to build and deploy applications consistently across different environments.

- **Dockerfile**: Docker uses Dockerfiles, which are text files containing instructions for building Docker images. Dockerfiles specify the base image, dependencies, environment variables, and commands needed to set up the application environment within the container.

- **Image Management**: Docker images are portable snapshots of containerized applications. They can be stored in registries like Docker Hub or private registries for easy distribution and sharing. Docker provides commands for building, tagging, pushing, pulling, and managing Docker images.

- **Container Orchestration**: Docker facilitates container orchestration, allowing users to manage and scale containerized applications across multiple hosts. Docker Swarm and Kubernetes are popular container orchestration tools that provide features like service discovery, load balancing, and automatic scaling for managing containerized workloads.

- **Microservices Architecture**: Docker promotes the adoption of microservices architecture, where applications are composed of small, loosely coupled services deployed as independent containers. Microservices improve scalability, flexibility, and maintainability by breaking down monolithic applications into smaller, manageable components.

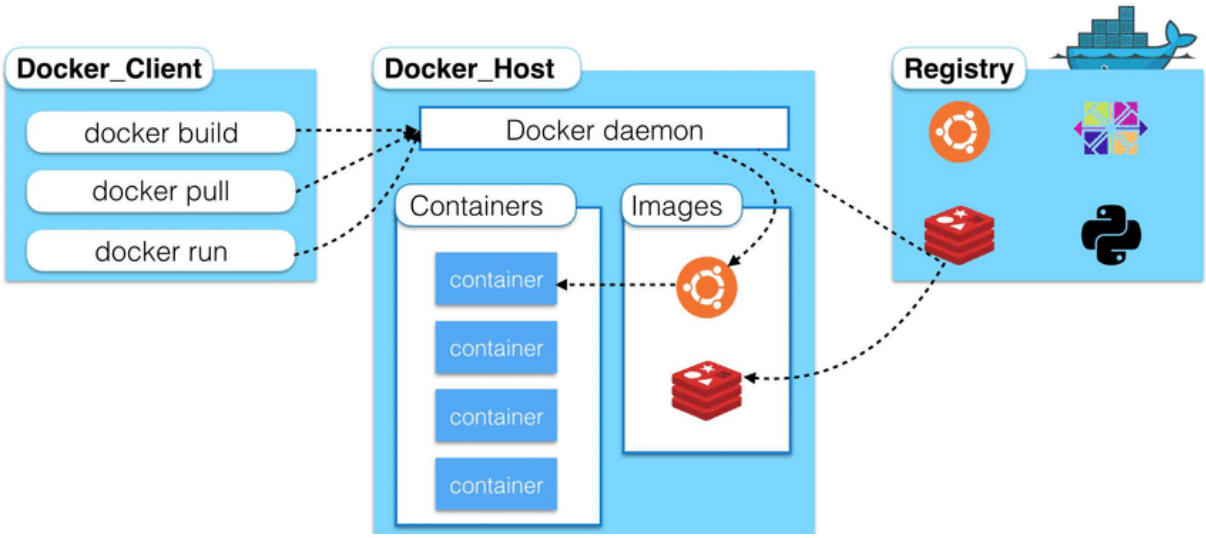

Figure 8. Internal architecture of Docker [50]

Frontend Development

REACT

React is a JavaScript library that have the main job of building user interfaces, developed and maintained by Facebook. It is widely used for building interactive and dynamic web applications, single-page applications, which stands for SPAs, and mobile applications. React follows a component-based architecture, where UI components are composed of reusable and independent components that manage their own abstraction state and rendering logic.
Implementing React in Software Development:

- **Component-Based Architecture**: React encourages the development of applications as a collection of reusable components. Components encapsulate UI elements and their behavior, allowing us to build complex components by composing smaller, self-contained and scalable components.
- **Virtual DOM**: React efficiently uses a virtual DOM (Document Object Model) to update the UI whenever a change event occurs within the application. Instead of directly manipulating the browser DOM, React updates a virtual representation of the DOM and computes the minimal set of changes needed to synchronize it with the actual DOM, resulting in improved performance and responsiveness.
- **Declarative Syntax**: React introduces a declarative syntax for designing ux/ui components, where developers describe what the UI should look like at any given time, rather than imperatively specifying how to update the UI in response to user interactions or changes in application state. This makes it easier to reason about UI behavior and maintain consistency across different components.
- **State Management**: React allows components to manage their own state using the **useState** hook or class-based state management. Stateful components can update their internal state in response to user interactions or asynchronous data fetching, triggering re-renders of the affected parts of the UI.

- **Component Lifecycle**: React components have a lifecycle that consists of various phases, such as mounting, updating, and unmounting. Developers can hook into these lifecycle events using lifecycle methods or the **useEffect** hook to perform tasks like data fetching, subscriptions, or cleanup operations [36-37].

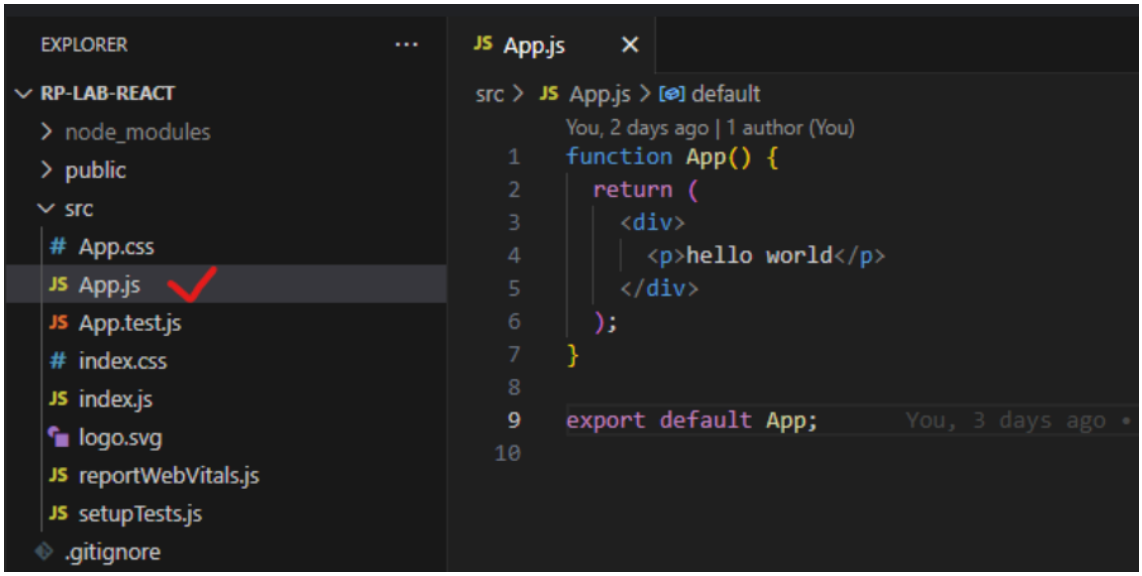

Figure 7. 'Hello World' in React JS Library

INTEGRATING REACT WITH BACKEND SERVICES

React is commonly integrated with backend services to create full-stack web applications. This integration involves making HTTP requests from React components to backend APIs to fetch and manipulate data. This enables React applications to interact with databases, authentication systems, and other backend services.

Key aspects of integrating React with backend services include:

- **API Integration**: React components use methods like **fetch** or libraries like Axios to make HTTP requests to backend APIs. These APIs handle business logic, database interactions, and other server-side tasks, returning data in JSON or other formats.
- **State Management**: React components manage UI state locally using React's state management features. Data fetched from backend services is often stored in component state, allowing components to render dynamically based on the fetched data.
- **Authentication**: Backend services often implement authentication mechanisms such as JWT (JSON Web Tokens) or session-based authentication. React applications interact with these authentication systems to authenticate users and manage user sessions.
- **Error Handling**: Integrating React with backend services requires robust error handling to deal with network failures, server errors, and other potential issues. React components display error messages to users and handle retries or fallbacks as needed [38-39].

These integration can be easily replicated and seen on the following figure.

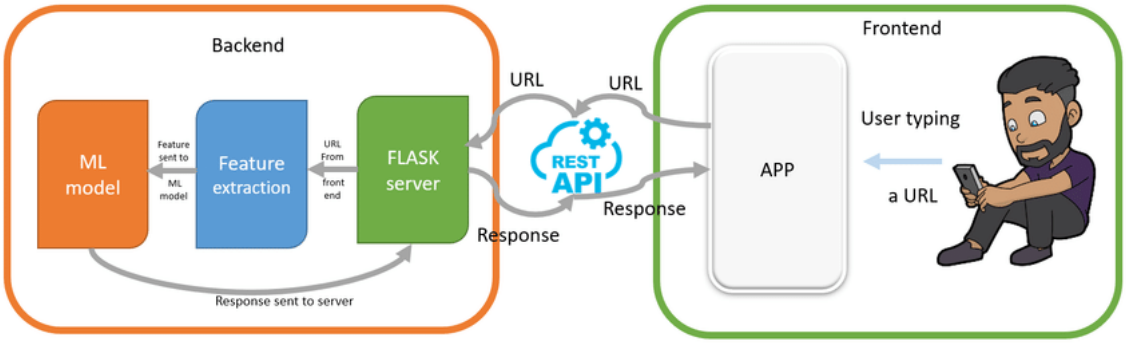

Figure 8. Frontend and backend communication throught http [51]

Software Development Methodologies

OVERVIEW OF AGILE METHODOLOGY

Agile methodology is an iterative approach to software development that emphasizes flexibility, collaboration, and customer feedback. It prioritizes delivering working software incrementally, with frequent iterations and continuous improvement throughout the development process [40].

There is some key aspects of Agile methodology, and some of them include:

- **Iterative Development**: Agile projects are divided into short iterations, typically sprints that have the duration of one to four weeks. Each iteration results in a working product

increment and added-value feature delivered, allowing for fast and consistent feedback and adaptation to changing requirements.

• **Collaborative Approach**: Agile teams consist of cross-functional members who work together to deliver value to a group of stakeholders in order to fulfill their needs. Collaboration between developers, testers, ux/ui professionals and stakeholders is essential for ensuring a global understanding of project goals and priorities within the team.

• **Customer Involvement**: Agile methodology emphasizes customer collaboration throughout the development process. Customers provide feedback on product increments, helping to validate requirements and ensure that the delivered software meets their needs.

• **Adaptability**: Agile projects embrace change and uncertainty, responding to new information and evolving requirements as they arise. Agile teams prioritize flexibility and adaptability, adjusting plans and priorities based on feedback and changing market conditions.

• **Continuous Improvement**: Agile teams regularly reflect on their processes and outcomes, seeking opportunities for improvement. Techniques like retrospectives and regular feedback sessions help teams identify strengths, weaknesses, and areas for growth. [41]

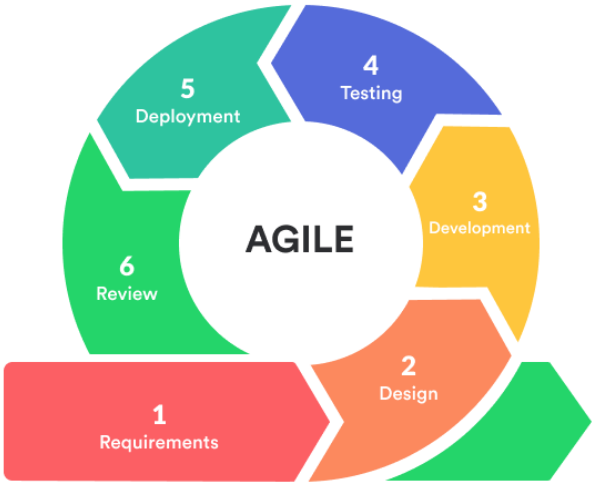

Figure 9. Agile Methodology Workflow [52]

The image depicts the Agile development methodology, a cyclical process that involves six key phases: Requirements, Design, Development, Testing, Deployment, and Review. This iterative approach allows for continuous improvement and flexibility throughout the project lifecycle.

Continuous Integration and Deployment (CI/CD)

Introduction to Jenkins CI/CD

Jenkins is a widely used open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) practices in software development. It automates the

process of building, testing, and deploying software, enabling teams to release high-quality code more frequently and reliably [42].
Some key aspects of Jenkins CI/CD include:

- **Continuous Integration (CI)**: CI is the practice of frequently integrating code changes into a shared repository, where automated builds and tests are triggered with each integration. Jenkins automates the CI process by monitoring version control systems for changes, executing build jobs, running tests, and providing feedback to developers.

- **Continuous Delivery (CD)**: CD extends CI by automating the deployment of code changes to production or staging environments after successful testing. Jenkins facilitates CD by orchestrating the deployment pipeline, which consists of multiple stages such as build, test, deploy, and release.

- **Pipeline as Code**: Jenkins Pipeline allows teams to define CI/CD workflows as code, stored alongside application code in version control repositories. Pipeline scripts specify the sequence of steps to be executed, including building, testing, and deploying the application, providing transparency and reproducibility.

- **Integration Ecosystem**: Jenkins integrates with a wide range of tools and technologies, including version control systems (e.g., Git, SVN), build tools (e.g., Maven, Gradle), testing frameworks (e.g., JUnit, Selenium), and cloud platforms (e.g., AWS, Azure). This integration ecosystem enables teams to customize and extend Jenkins to suit their specific requirements.

- **Scalability and Flexibility**: Jenkins is highly scalable and can be deployed in various environments, including on-premises servers, cloud infrastructure, and containerized environments. It supports distributed builds and parallel execution, allowing teams to handle large-scale projects and improve build times.

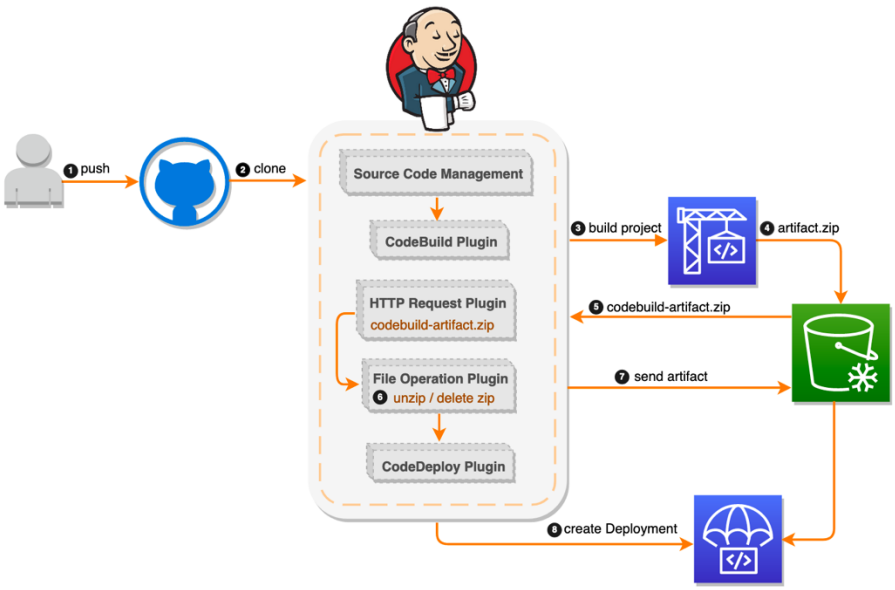

Figure 10. Jenkins Base Architecture and Workflow [53]

The previous image displays the potentiality and integrations of using Jenkins in deployment automation. Integrating with with git and nexus, we can build packages and run some operations by demand to achieve automation.

Platform as a Service (PaaS)

Overview of PaaS

Platform as a Service (PaaS) is a cloud computing model that offers a platform for developing, deploying, and managing applications without the need to manage the underlying infrastructure.
There are some key features on it, which include [43-44]:

- Abstraction of infrastructure.
- Development and deployment tools.
- Scalability and elasticity.
- Managed services.

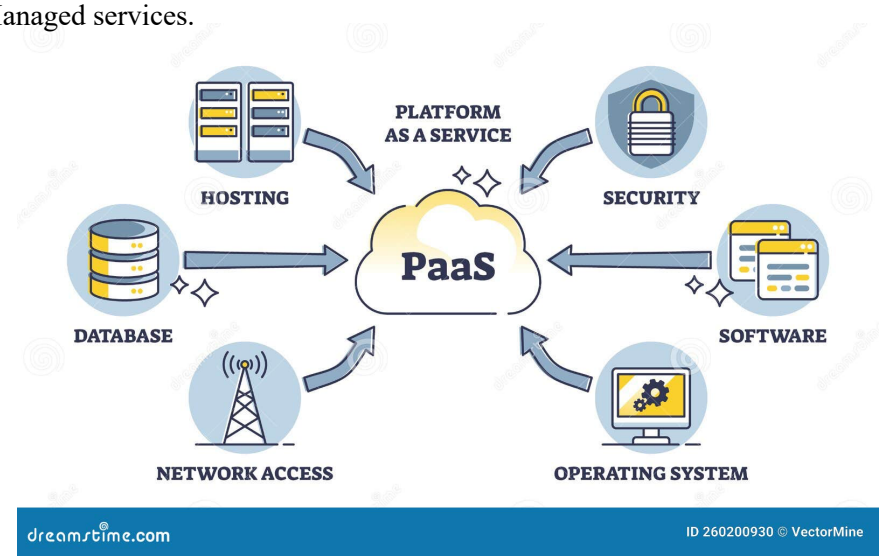

Figure 11. Common real-world Example of Platform As a Service [54]

Application Monitoring and Documentation

Application monitoring and documentation are critical aspects of maintaining the health and performance of software systems. In modern software architectures, continuous monitoring, real-time data analysis, and thorough documentation are essential for diagnosing issues, optimizing performance, and ensuring reliability.

Grafana

Grafana is a widely-used open-source platform for monitoring and observability. It provides powerful and flexible dashboards for visualizing time-series data from various data sources. In this architecture, Grafana serves as the primary tool for visualizing metrics and logs collected from applications.

- **Data Visualization**: Grafana is used to create dynamic and interactive dashboards that allow users to explore metrics and log data in real time.
- **Alerts and Notifications**: Custom alerts can be set up within Grafana to notify administrators of critical issues based on the data trends.

- **Plugins and Integrations**: Grafana supports numerous plugins that allow for extended functionality, including various visualization types, integrations with other monitoring tools, and enhanced data management.

Figure 12. Example of Grafana Dashboard

INFLUXDB

InfluxDB is a high-performance time-series database optimized for handling large volumes of time-stamped data. It is particularly suited for storing metrics and log data due to its efficient storage and querying capabilities.

- **Metrics Storage**: All application performance metrics, such as CPU usage, memory utilization, and request rates, are stored in InfluxDB.
- **Log Data Storage**: In addition to metrics, logs generated by the applications are also stored in InfluxDB. This enables the correlation of logs with metrics for comprehensive monitoring.
- **Data Retention and Compression**: InfluxDB allows for customizable data retention policies and efficient data compression, ensuring that historical data can be stored for long-term analysis without excessive storage costs.

LOGGING

Logging is a critical component of application monitoring. Logs provide detailed insights into the internal workings of applications, recording events, errors, and other significant activities.

- **Centralized Log Management**: All application logs are collected and stored in InfluxDB, allowing for centralized management and analysis.
- **Structured Logging**: Logs are stored in a structured format, enabling efficient querying and analysis. This structured approach supports the quick identification of patterns and anomalies.
- **Integration with Metrics**: Logs are closely integrated with metrics in the monitoring setup, allowing for a unified view of application behavior.

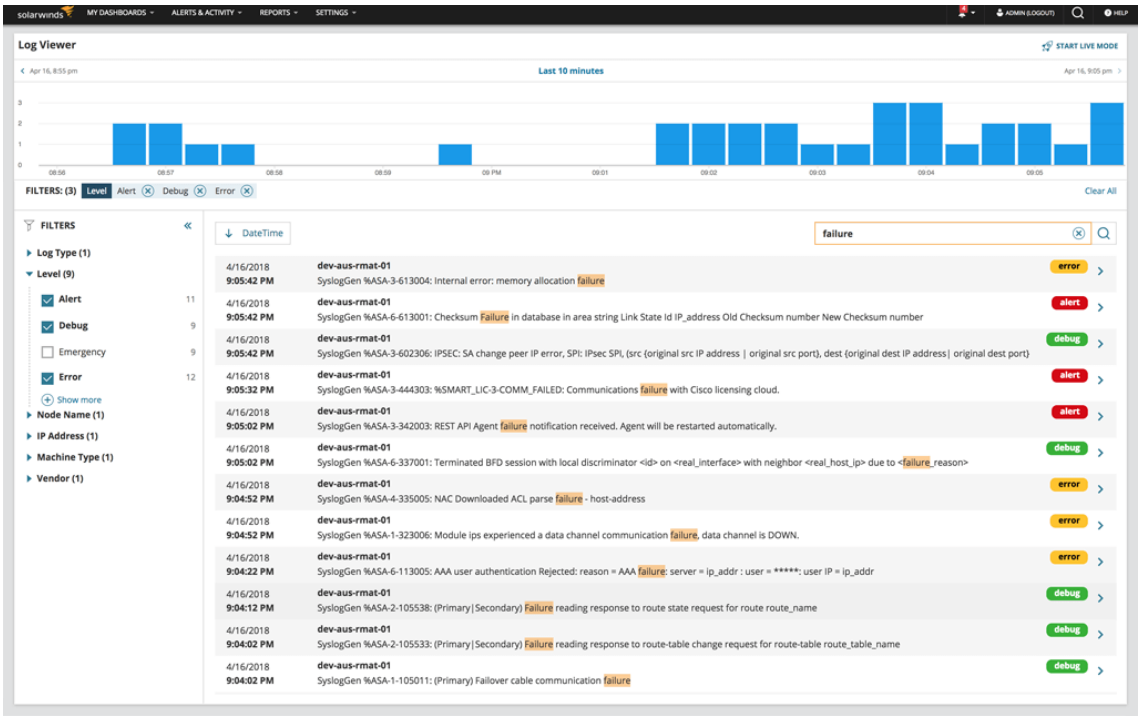

Figure 13. Logging example and it's importance

METRICS

Metrics provide quantitative data about the performance and health of applications. They are essential for tracking system behavior over time and identifying performance trends.

- **Performance Monitoring**: Metrics related to application performance (e.g., response times, throughput) are continuously collected and stored in InfluxDB.
- **Resource Usage**: Metrics tracking system resource usage (e.g., CPU, memory, disk I/O) are monitored to ensure the efficient operation of the application infrastructure.
- **Dashboards**: Grafana is used to create real-time dashboards that visualize these metrics, providing immediate insights into the system's status.

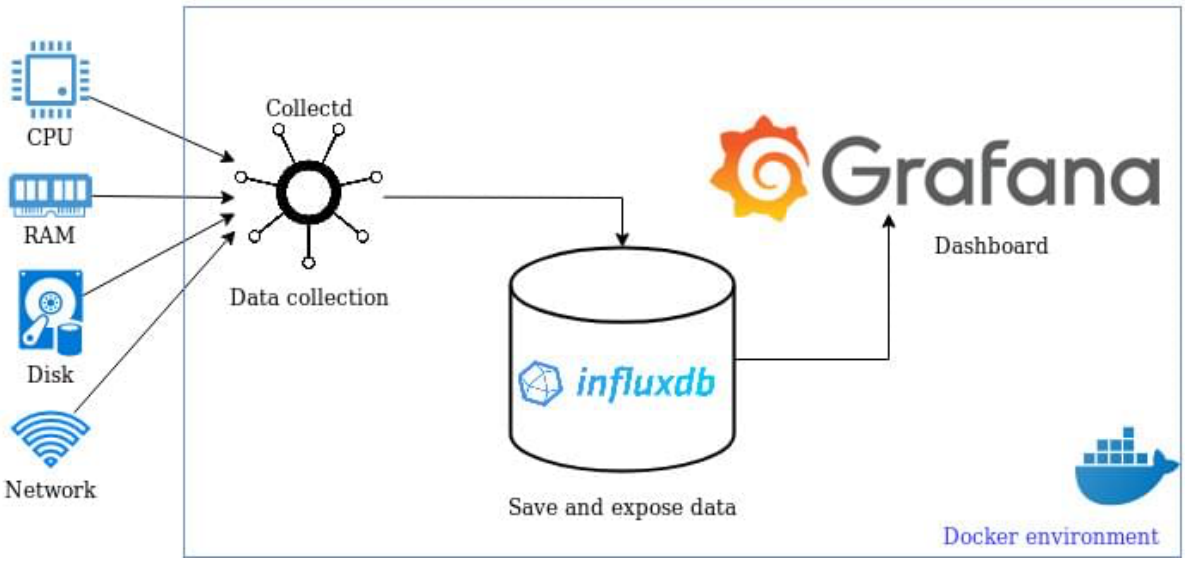

Figure 15. InfluxDB and Grafana Integration

Open Source

Open source refers to software with source code that is made publicly available, allowing anyone to view, modify, and distribute it. This approach promotes transparency, community-driven development, and collective problem-solving.

Open source software (OSS) thrives on the contributions of developers worldwide, fostering rapid innovation and adaptability. Popularized by projects like Linux, Git, and Apache, the open source model has revolutionized the tech industry, empowering organizations to build upon shared knowledge while reducing costs.

Beyond software, the open source philosophy extends to hardware, data, and even research, accelerating technological progress across sectors.

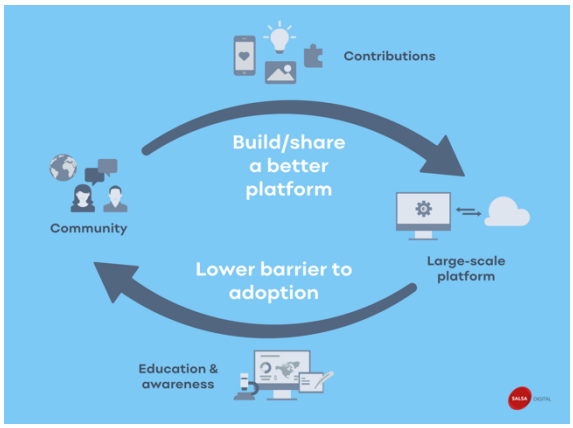

Figure 16. Open Source community contributions workflow

Implications of the State-of-the-Art Technologies

The implications of state-of-the-art technologies extend beyond their immediate technical benefits to reshape how businesses and teams operate. For instance, the adoption of Agile methodologies has led to a cultural shift within organizations, emphasizing collaboration, transparency, and iterative progress. Agile promotes continuous feedback loops, enabling teams to adapt to market demands and user needs more effectively. Similarly, tools like Git and Jenkins facilitate streamlined collaboration and automation, ensuring that teams can manage version control, perform code reviews, and deploy applications with unparalleled efficiency. These advancements collectively reduce time-to-market while maintaining high-quality software standards.

Furthermore, technologies like Docker and PaaS have profound implications for scalability and resource optimization. Docker's containerization allows applications to be deployed consistently across multiple environments, eliminating the common "it works on my machine" problem and enhancing portability. PaaS solutions abstract infrastructure management, enabling developers to focus solely on building applications while ensuring scalability and reliability. Combined with authentication technologies, these tools ensure that applications are not only efficient but also secure, meeting the increasing demands for cybersecurity in a rapidly evolving digital landscape. Together, these technologies empower organizations to innovate, scale, and deliver software solutions that address both current and future challenges effectively.

Future Trends and Research Directions

Future trends and research directions in the realm of state-of-the-art technologies encompass advancements in areas such as cloud computing, microservices architecture, artificial intelligence, machine learning, and cybersecurity. Research endeavors may focus on optimizing existing technologies, exploring emerging paradigms, addressing scalability challenges, enhancing developer productivity, and ensuring the security and reliability of software systems.

In conclusion, by leveraging technologies such as Golang, HTTP protocol, REST principles, Git, Docker, React, Agile methodology, Jenkins CI/CD, and PaaS, the Appjet project can effectively automate manual processes, simplify deployment workflows, and accelerate deployment cycles. These technologies enable faster development and deployment cycles, improved collaboration, enhanced security, and scalability, aligning with the goals of Appjet to streamline server configuration processes and deployment of web applications. As the project progresses, further exploration of emerging trends and research directions in software development will be essential to maintain competitiveness and drive innovation in the ever-evolving digital landscape.

Competitors Analysis

This section content was compiled based on results retrieved on SemRush and similarweb Competitors Analysis websites. (https://www.semrush.com/projects/, https://pro.similarweb.com/)

Competitors Detection/Identification

To identify competitors for Appjet, we focused on understanding its core functionalities and target market. Appjet's primary goal of simplifying server configuration and web application deployment through automation provided a clear lens for competitor analysis. We began by researching platforms with overlapping objectives, such as simplifying deployment workflows, offering automation tools, or supporting collaborative development. This involved keyword searches for deployment automation, exploring web for similar tools, and leveraging industry resources like Crunchbase and GPT to identify potential competitors. Additionally, analyzing the preferences and challenges faced by Appjet's target audience helped pinpoint platforms addressing similar user needs.

Next, we evaluated potential competitors based on feature similarity, market position, and user feedback. Platforms with functionalities like server monitoring, configuration management, automated deployments and integration support were prioritized as direct competitors, while broader tools offering partial overlap were classified as indirect competitors. A detailed analysis of user reviews, pricing models, and unique offerings helped refine the list to the top five competitors. This process ensured that Appjet's positioning could address market gaps, differentiate its value proposition, and maximize its impact on the deployment automation ecosystem.

Overview Statistics

When we were in the brainstorm process, and got the idea of Appjet, first we needed to know which platforms do the same in a similar way. So, this chapter is reflecting the exact study we did prior to the start of the development cycle off Appjet. The following table represents the top five potential competitors of Appjet.

Table 1. Top five Competitors Analysis comparison

#	Company	Website	Visits (Last Month)	Time on site	Pages per visit	Bounce Rate	Global Rank	Organic vs Paid Traffic	Top3 Traffic Sources
1	Vercel	https://vercel.com	7,930,000	00:04:02	3.25	45.32%	#8,294	80% vs 20%	Direct / Search / Social
2	Netlify	https://netlify.com	5,700,000	00:03:15	3.10	50.12%	#12,103	85% vs 15%	Search / Direct / Referral
3	Heroku	https://heroku.com	4,200,000	00:02:58	2.90	55.14%	#15,542	82% vs 18%	Search / Direct / Social
4	Render	https://render.com	2,800,000	00:03:45	3.40	40.25%	#21,110	78% vs 22%	Direct / Search / Referral
5	Fly.io	https://fly.io	1,500,000	00:03:10	3.00	49.50%	#28,645	70% vs 30%	Search / Direct / Social

Each Column is defined and have it's own meaning. We can define them as:

- **#:** The index number (the higher the better)
- **Company**: The name of the company providing web application deployment services.
- **Website**: The URL of the company's official website.
- **Visits (Last Month)**: The total number of visits to the website in the previous month. This metric helps gauge the website's popularity and user engagement.
- **Time on site**: The average duration a visitor spends on the website during a single session. This indicates how engaging the website content is to users.
- **Pages per visit**: The average number of pages viewed by a visitor during a single session. This shows how deeply users are interacting with the website.
- **Bounce Rate**: The percentage of visitors who leave the site after viewing only one page. A lower bounce rate typically indicates that visitors find the site relevant and are exploring it further.
- **Global Rank**: The global ranking of the website based on traffic and engagement metrics compared to all other websites. A lower rank number indicates higher popularity.
- **Organic vs Paid Traffic**: The percentage breakdown of website traffic that comes from organic search (unpaid) versus paid advertising. This helps understand how much of the site's traffic is earned versus bought.
- **Top 3 Traffic Sources**: The three main channels driving traffic to the website, such as direct visits (typing the URL directly), search engines, social media, or referrals from other websites. This shows where the site's audience is coming from.

Comparing some competitors

The following data, reflects the Month of July 2024. To have a more extensive and detailed analysis we must have and pay for the premium service of the similarweb website.
We will use vencel, netflify and fly.io companies.

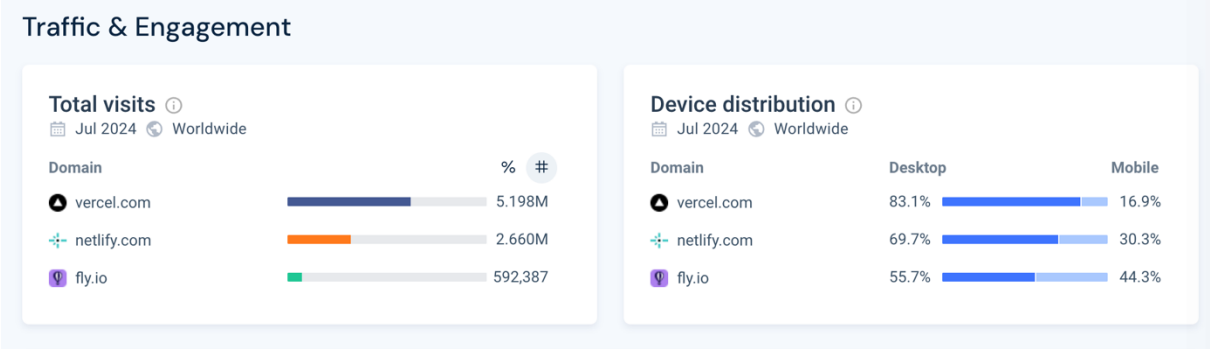

Figure 17. Competitors Traffic & Engagement

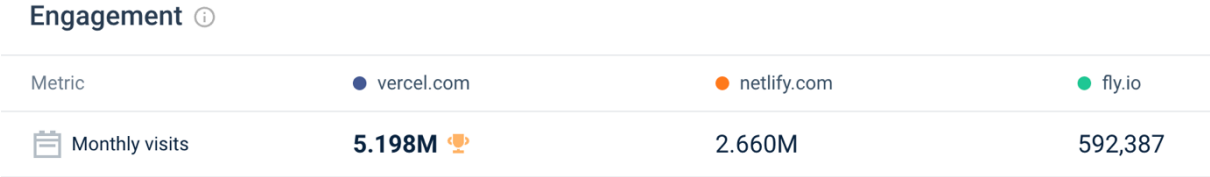

Figure 18. Total Absolute Monthly visits per competitor on Top three Competitors

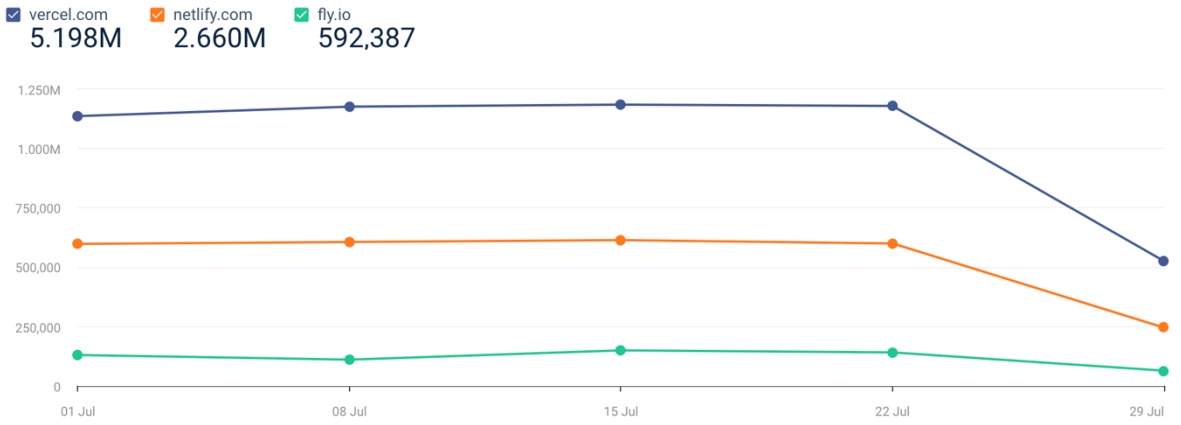

Figure 19. Total Evolution in July of 2024 visits per competitor on Top three Competitors

Some Top Organic Keywords

In this section we will use Vercel, Netflify and Fly.io companies in order to know which are the most used organic keywords for each company.

An organic keyword is a word or phrase used in online search queries that helps users find relevant content naturally, without paid advertisements. When a website ranks well for organic keywords, it appears higher in search engine results due to its relevance and quality, rather than because the site has paid for the placement. Organic keywords are essential for SEO (Search Engine Optimization) to drive free, long-term traffic to a website.

Fly.io

- **fly io** - High search volume keyword directly related to the brand.
- **fly hosting** - Common keyword for users searching for hosting services provided by Fly.io.
- **fly.io pricing** - Users interested in Fly.io's pricing structure.
- **docker hosting** - Related to Fly.io's ability to host Docker containers.
- **serverless docker** - Reflecting interest in Fly.io's serverless Docker hosting solutions.

Vercel

- **vercel** - Brand-specific keyword with high organic traffic.
- **vercel pricing** - Users searching for details on Vercel’s pricing plans.
- **next.js** - Strongly associated with Vercel due to its ownership of the Next.js framework.
- **deploy with vercel** - Users looking for deployment options on Vercel.
- **serverless functions** - Tied to Vercel's serverless functions offerings.

Netlify

- **netlify** - Primary brand keyword.
- **netlify cms** - Popular content management solution provided by Netlify.
- **jamstack** - Reflecting Netlify's strong association with JAMstack architecture.
- **netlify deploy** - Users searching for how to deploy projects using Netlify.
- **netlify forms** - Related to the forms handling feature of Netlify.

APPJET Advantages VS Competitors

User Experience and Interface

- **Vercel**: Known for its seamless integration with Next.js and a strong focus on frontend deployment. However, its CLI and UI interfaces are limited in their customization and flexibility.
- **Netlify**: Offers a robust UI and CLI but may require more configuration for advanced use cases. It excels in JAMstack applications but can be complex for users needing more customization.
- **Fly.io**: Focuses on containerized applications with a strong emphasis on global deployment. Its UI and CLI are functional but geared primarily towards developers with container expertise.

- **Heroku**: Provides a comprehensive platform with a straightforward UI and CLI. However, its cost structure and performance can be limiting for scaling applications.
- **Render**: Offers a good balance of UI and CLI functionalities but lacks some advanced features found in more specialized platforms.

Appjet Advantage: Appjet's unique combination of CLI and UI applications ensures a highly customizable and user-friendly experience. The CLI provides powerful, command-line-based configuration and deployment capabilities, while the UI application offers a streamlined interface for users who prefer graphical interaction. The Appjet Server Daemon adds another layer of control, managing deployment and application health efficiently.

Deployment and Flexibility

- **Vercel**: Excellent for frontend frameworks like Next.js but less flexible for backend or custom setups. Deployment options are somewhat constrained to specific frameworks.
- **Netlify**: Highly flexible for static and JAMstack sites but less optimized for complex backend services or custom environments.
- **Fly.io**: Provides extensive support for Docker and serverless solutions, but may be overkill for simpler use cases and requires specific expertise.
- **Heroku**: Known for its broad support for various programming languages and frameworks but can become expensive and less efficient for scaling.
- **Render**: Offers a solid range of deployment options but lacks the deep customization available in more specialized tools.

Appjet Advantage: Appjet's architecture is designed for versatility. The Appjet CLI supports a wide range of configurations and deployment strategies, while the Server Daemon ensures that applications are monitored and managed efficiently across different environments. This flexibility makes Appjet suitable for a broader range of use cases compared to more specialized competitors.

Performance and Monitoring

- **Vercel**: Provides robust performance metrics and monitoring but is highly tailored to frontend applications.
- **Netlify**: Offers performance monitoring with a focus on static and JAMstack sites, but can be limited for dynamic applications.
- **Fly.io**: Strong in performance, especially for containerized applications with global distribution. However, its monitoring tools may not be as intuitive for all users.
- **Heroku**: Provides comprehensive monitoring and performance tools but can struggle with performance at scale and has a complex cost structure.
- **Render**: Good performance monitoring but less advanced than some competitors in terms of scaling and optimization.

Appjet Advantage: Appjet's Server Daemon is specifically designed for comprehensive performance monitoring and management. It continuously monitors application health and resource usage, providing detailed insights and automatic adjustments to ensure optimal performance. This level of monitoring is superior to what is available from many competitors, making Appjet an ideal choice for users needing reliable and responsive application management.

Cost Efficiency

- **Vercel**: Offers a free tier with basic features but can become costly as usage scales, especially for advanced features.

- **Netlify**: Has a competitive pricing model but may become expensive for high-traffic or complex sites.
- **Fly.io**: Provides flexible pricing based on usage but may involve higher costs for extensive global deployments and containerized services.
- **Heroku**: Known for its high cost, especially when scaling applications and using add-ons.
- **Render**: Generally more affordable but still can become costly depending on the deployment needs and resource usage.

Appjet Advantage: Appjet aims to provide a FREE and Open Source solution and low cpu cost by optimizing resource allocation and deployment strategies. Its architecture allows for efficient scaling without the high costs often associated with other platforms. Additionally, the flexibility of the CLI and Server Daemon ensures that users only need to use the executables they need, with no hidden costs or unnecessary overhead.

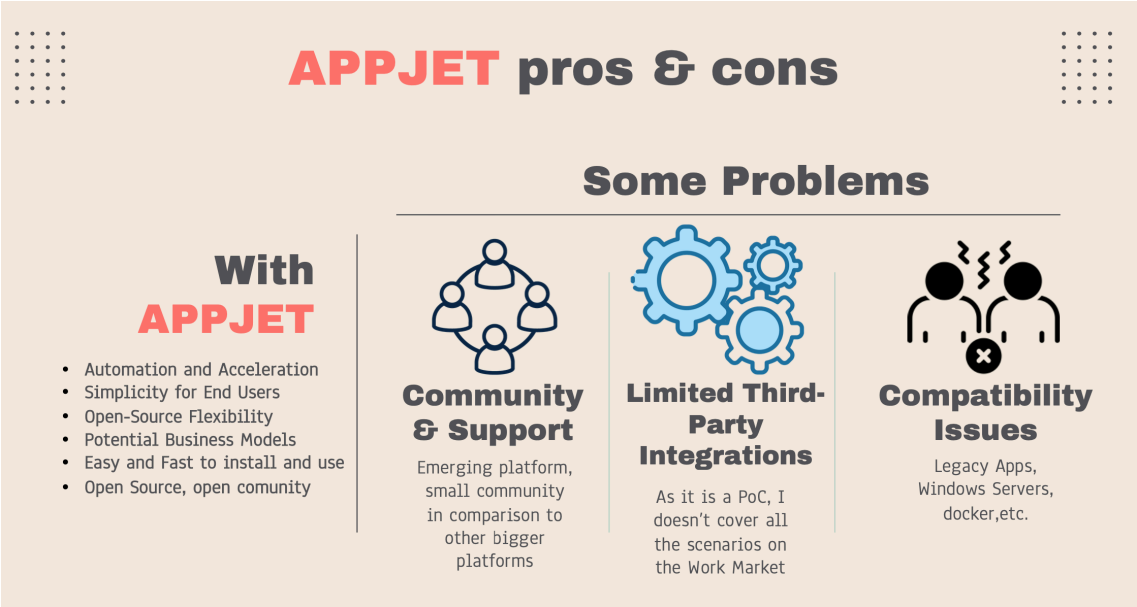

Figure 20. Some Appjet's Pros and Cons

Chapter 3. Development

Concept

This project is divided into three parts. The first component is Appjet CLI, a command-line application designed to run on the client side as a console application. Also, another component to run on client side but as UI application, not console application.

This CLI provides a user-friendly interface for interacting with the Appjet platform, allowing for seamless configuration and deployment of web applications. Lastly, the project includes Appjet Server Daemon, which defines the executable that will run on each target server where applications will be deployed.

This daemon is responsible for managing the deployment process, monitoring application health, and ensuring smooth operation of deployed applications. Throughout this chapter, it will mention the technical implementation of each component, detailing the tools, technologies, and methodologies employed in their development.

By dissecting the inner workings of each part, we aim to provide a comprehensive understanding of how the Appjet platform operates and how it fulfills its goal of simplifying server configuration processes and streamlining the deployment of web applications, in conclusion this module serves as the backbone of the Appjet platform, managing decisions related to deployment strategies, resource allocation, and system optimization.

The source code of Appjet, as it is open source, it is available at Github Repository. (https://github.com/guilhermemalhado1/Appjet)

Requirements Analysis

In this report section, we will be explaining and reflecting on the Requirement Analysis, which is the process that intercept the customer needs and software requirements. This analysis consists essentially of data and requirements gathering to fulfill the need involved, developing a solution.

The requirements are divided into two main types: Functional and Non-Functional. In one hand the Functional Requirements usually refer to which function the system should have, some functionality that the system must implement and have. In Other hand, the Non-Functional Requirements, serve as criteria which qualify and complement the functional requirements. In the following sub-chapters, we will be discussing these two types of requirements, in a more detailed approach.

Some key points of this requirement Analysis is divided in two main parts, such as Stake Holders as first segment, which in this case is Professor Doctor João Ventura and Me or any other person that can benefit from this solution to get his problem solved.

As second segment, High-Level overview of requirements, which we will deep-dive in the following two chapters and at last but not least the methodology, which was some sort of Waterfall model mixed with Agile's Kanban framework, where we only started designing the platform, after getting the requirements done based on real stake holders feedback and needs.

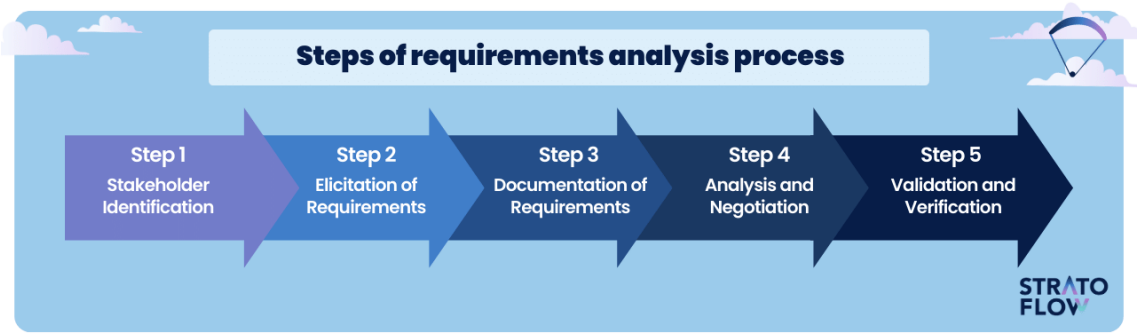

Figure 21. Requirements Analysis workflow

Functional Requirements

In the next bullet points, it will be pointed the main functional requirements of Appjet, so we can have a full overview as big picture view of Appjet.

- REQ.1: Develop Appjet Platform
  - REQ.1.1: Develop Appjet Server Daemon
    - REQ.1.1.1: The system must serve static project documentation
    - REQ.1.1.2: The system must be able to deploy web applications without any configuration needed only by receiving a configuration file
    - REQ.1.1.3: The system must provide containers status on demand when requested
    - REQ.1.1.4: The system must have logging for troubleshooting
    - REQ.1.1.5: The system must have metrics
    - REQ.1.1.6: The system must provide endpoints for starting, restarting or even disabling the application temporarily.

        - REQ.1.1.7: The system must have token authorization and authentication as an extra security layer
    - REQ.1.2: Develop Appjet CLI (Client Command Line Interface)
        - REQ.1.2.1: The system must be able to provide artifact to running in both linux or windows operative systems.
        - REQ.1.2.2: The system must be able to customize a configuration file
        - REQ.1.2.3: The system must be able to send the config file to the server
        - REQ.1.2.4: The system must allow the user to send commands to the server (Such as deploy, restart, stop, start, etc.)
        - REQ.1.2.5: The system must allow a user to log in and retrieve authorization tokens.
        - REQ.1.2.6: The system must be able to run itself in UI mode.
    - REQ.1.3: Develop Appjet Client UI App
        - REQ.1.3.1: The system must replicate all of the requirements present in the previous 1.2 Requirement, but on UI interfaces in spite of Console interfaces.

Non-Functional Requirements

In the next bullet points, it will be pointed the main non-functional requirements of Appjet, so we can have a full overview as big picture view of Appjet.

- REQ.1: The system must be of easily and fast maintenance
- REQ.2: The UI on client-side application must be user friendly
- REQ.3: The system must be scalable and generic, so new functions can be added later on without break the already existent functionalities
- REQ.4: The software must be open source to have public contributions by the community
- REQ.5: The system must integrate InfluxDB and Grafana

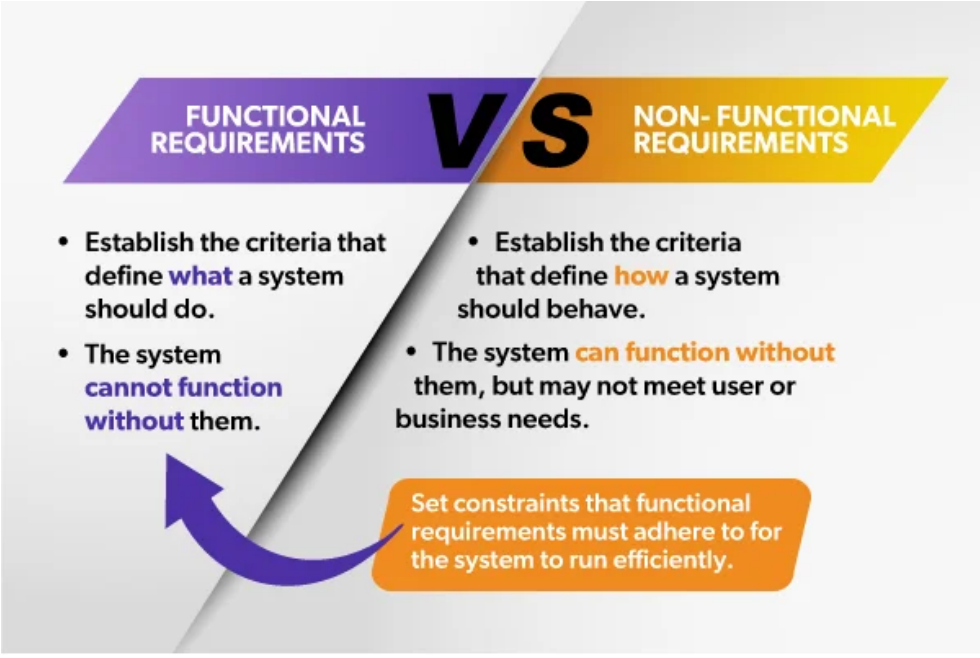

Figure 22. Functional and Non-Functional Requirements Comparison [55]

Functional Requirements Diagram

The following diagram displays visually on a diagram, the functional requirements.

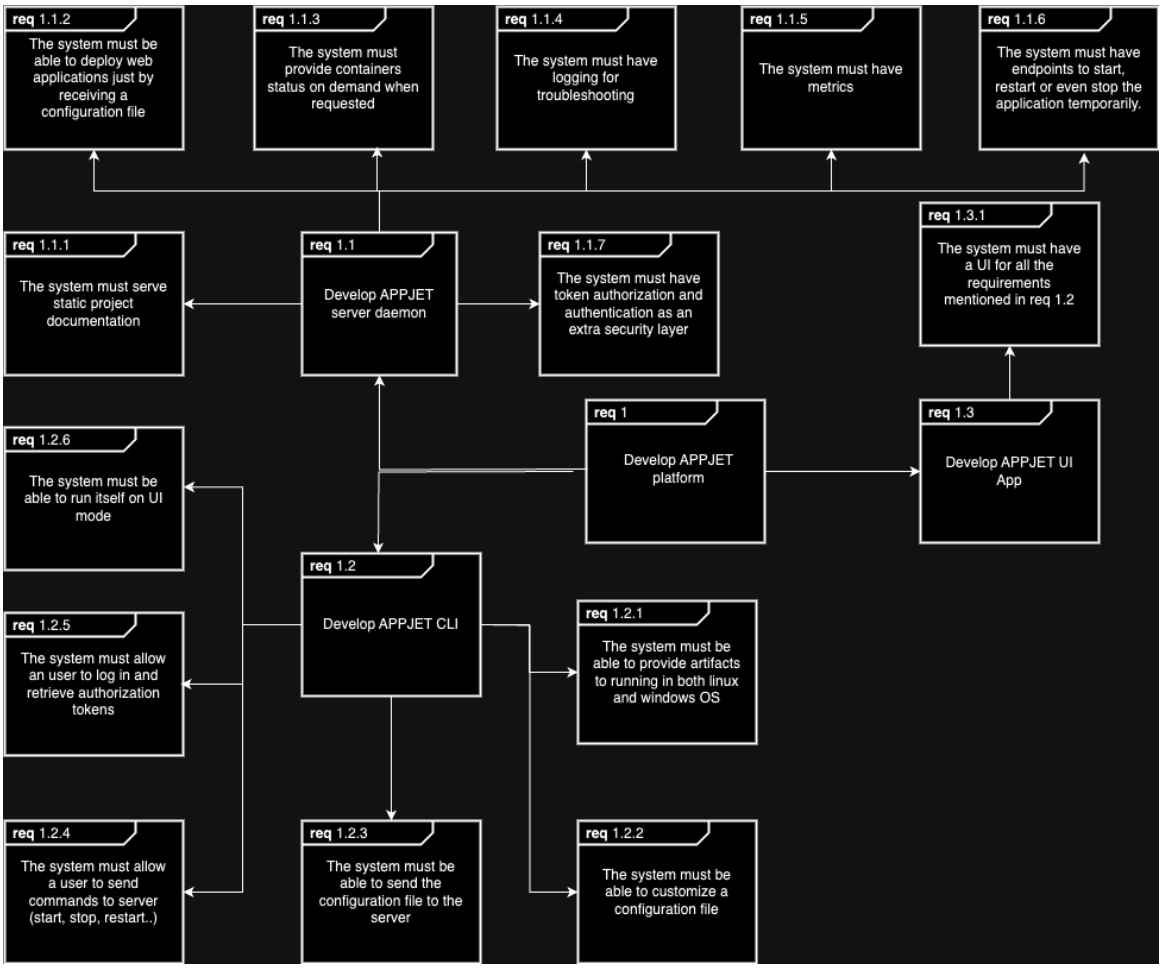

Figure 23. Functional Requirements Diagram

Use Cases

As any other platform, we needed to identify the main use-cases before we start developing. So we have strict requirements gathered for the current proof of concept, Appjet.

This section will point the main use cases of the current state of the application.

- Deploy Custom Web Application
- Restart Application
- Deploy a new version of the Application
- Check the Deployed Application configuration file on a Specific Server
- Checking Application Status
- Checking Application Metrics
- Monitor Application Logging
- Consult Application Documentation

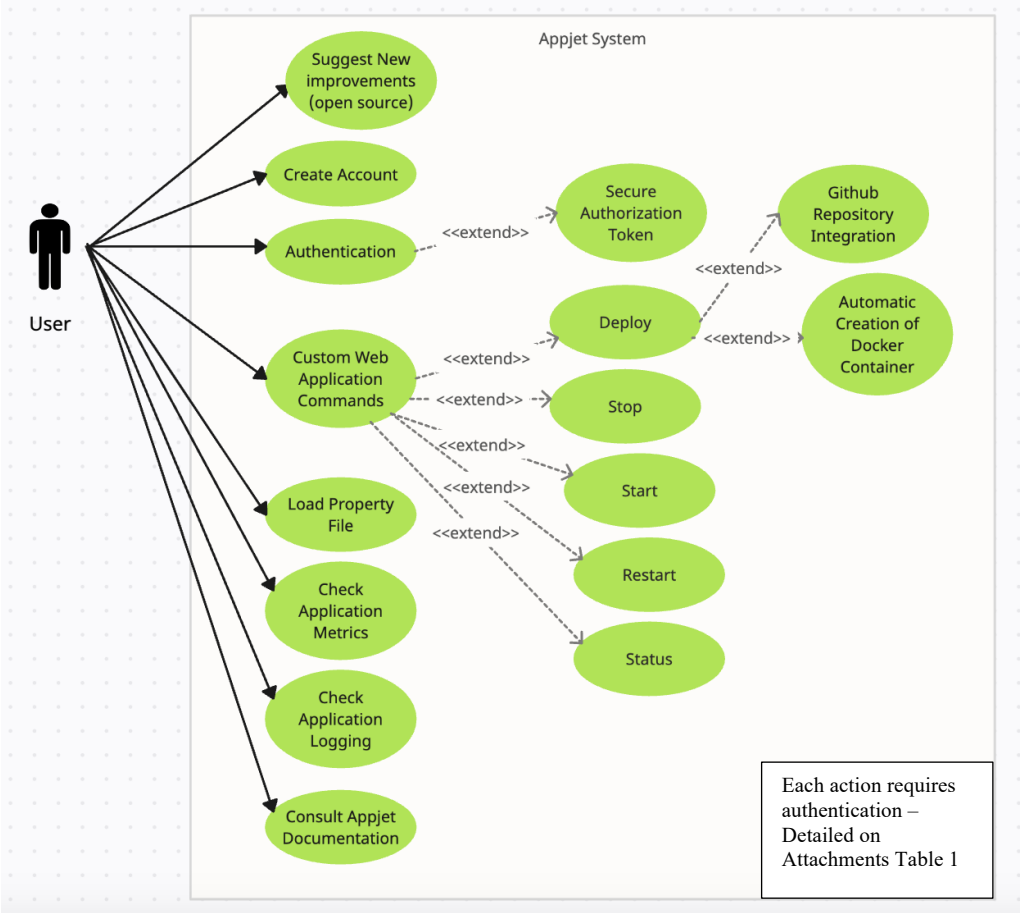

Figure 24. Main Use Cases available at the moment on Appjet

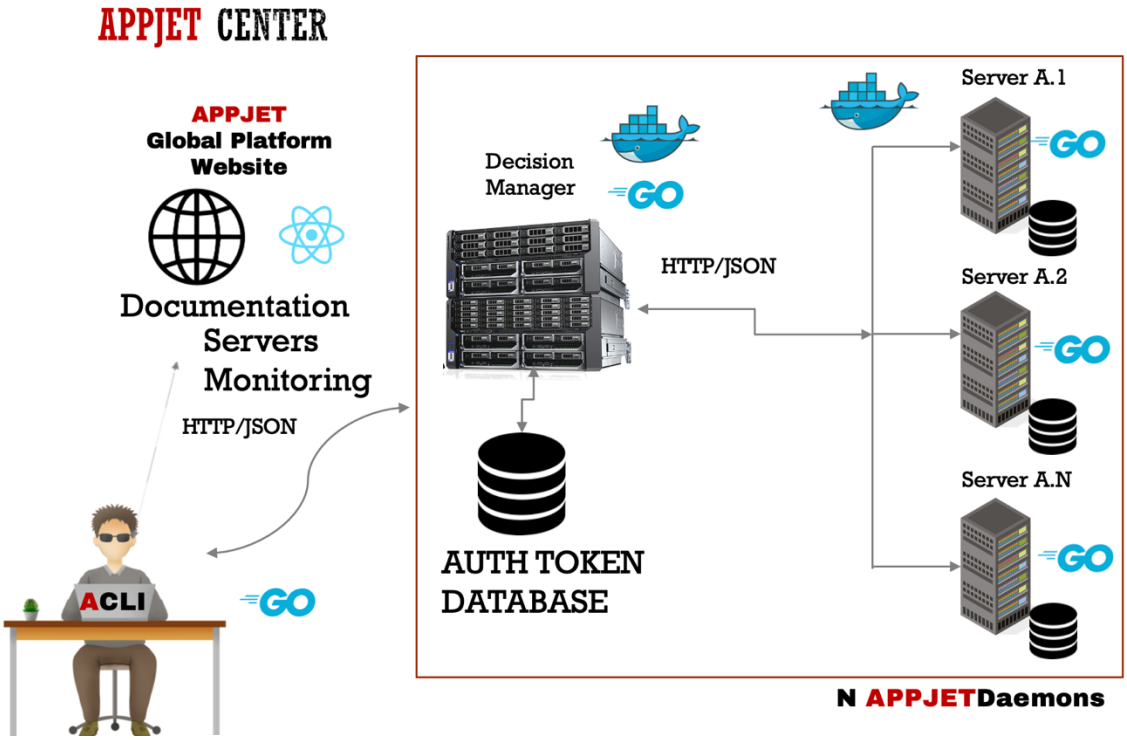

Figure 25. Real High Level Appjet Architecture

The previous image, represents an high level architecture diagram of Appjet.
The image illustrates the high-level architecture of Appjet, a platform that manages a group of N AppjetDaemons. These daemons are responsible for executing tasks and can be deployed on various servers (Server A.1, Server A.2, ..., Server A.N). The Appjet Center, accessible through the Global Platform Website, provides a central point for managing and monitoring the entire system. It includes components such as the Decision Manager, which orchestrates task execution, and the Documentation and Monitoring servers, which offer insights into the platform's operations. Communication between components primarily occurs through HTTP/JSON requests, ensuring interoperability. The architecture also incorporates an authentication mechanism using an AUTH TOKEN DATABASE, which manages user access control. The CLI (Command-Line Interface) provides a way for users to interact with the platform and manage tasks directly.

Configuration File – The core of Appjet

Following a Configuration-first logic, Appjet server daemon decision making is based on a uploaded configuration file. Which will dictate the behavior and the business logic that will be dynamically done to assure the desired objective.

```
{
    "app_name": "app_name_xpto",
    "language": "python",
    "github_repo": "<github-clone-repo-http-url>",
    "github_user": "<github-username>",
    "github_password": "<gibthub-password>",
    "docker_image": "python:3.9-slim",
    "target_urls": [
        {
            "url": "http://localhost:8081",
            "name": "local-server"
        }
    ]
}
```

Figure 26. Appjet Configuration file

This file basically is the core and input of the Appjet Platform as shown on the diagram specified on the next page, on Figure 23. Here is where the heart of the platform is – without this structure, nothing will work.

- app_name: Represents the desired name of the docker container where the target web app will be deployed.
- language: Represents the language that the web app is written on
- github_repo: Represents the URL of the github repository where is the source code of the web app
- github_user & github_password: Represents the user credentials of the git account with permissions to that repository

- docker_image: Represents the name of the docker image that will be used behind the scenes (Needs to exist in dockerhub)
- target_urls: Consists in an array of Servers. Each server is where the web app will be deployed. A server have a name and a URL.

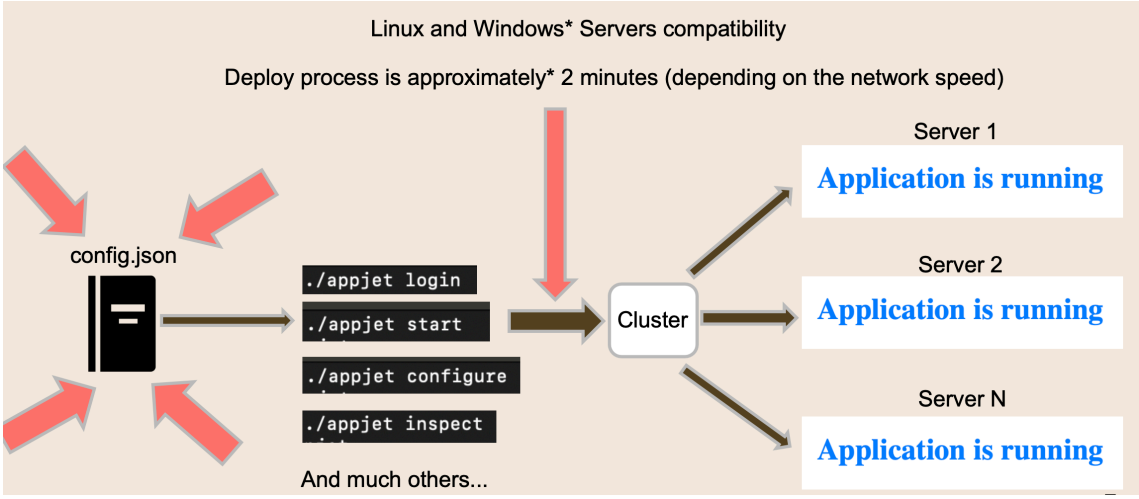

Figure 27. Configuration file as Automation Trigger on Appjet

Right now, the JSON configuration file needs to exactly match the template structure, and it needs to be fulfilled automatically by the user that wants to use APPJET. But in the future, we will have some AI tool that will analyze the project directory or git remote repository project, and automatically fill the config.json file, so the user doesn't need to interact with it.

This image shows the login, start, configure and inspect command, in the Attachments section there is a table with all of the available commands at the moment of this document submission.

Architecture and Components Diagram

This following diagram outlines the architecture of a system that integrates various components, each serving a distinct role within the overall environment. This system is designed with modularity in mind, using Docker containers to isolate and manage various services while ensuring they can communicate effectively through well-defined interfaces. The use of Go for packaging the Appjet components indicates a focus on performance and efficiency, with robust monitoring tools in place to maintain system health. The architecture supports scalability, maintainability, and easy deployment across different environments.

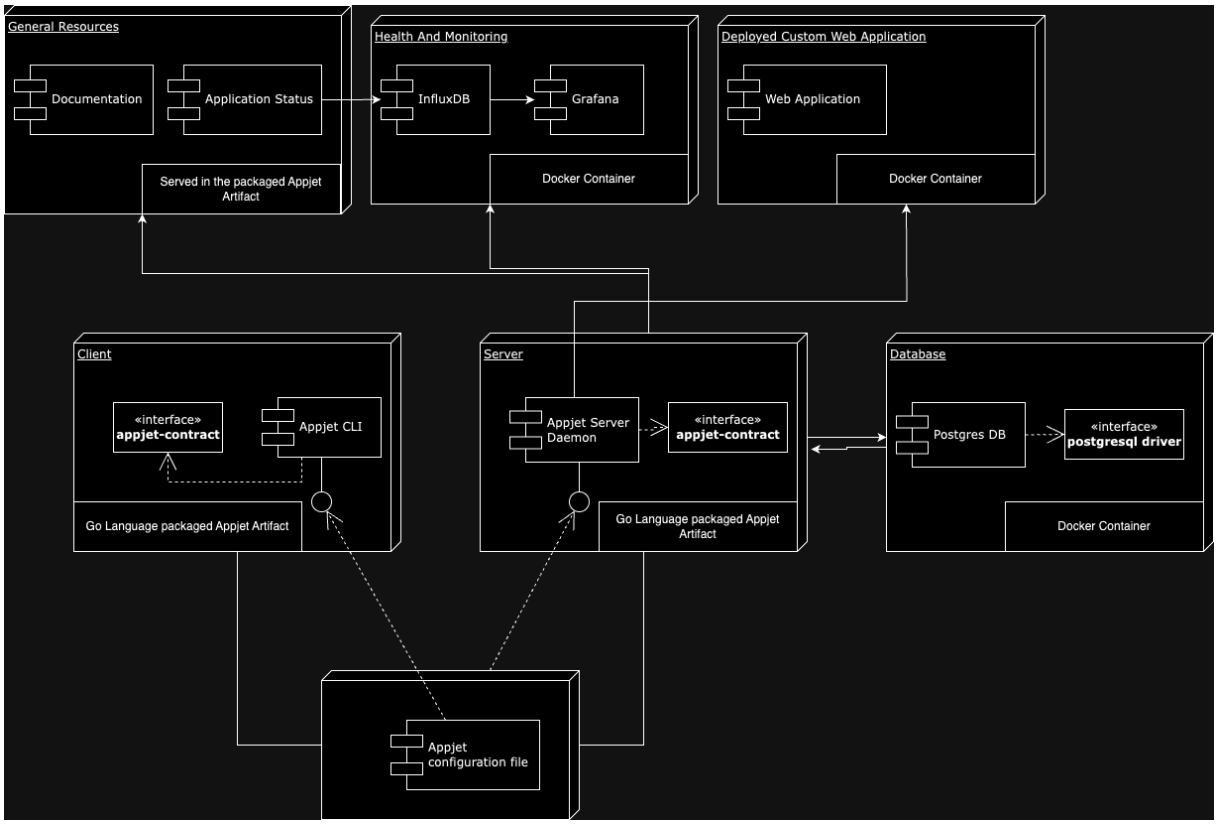

Figure 28. Appjet Architecture and Components Diagram

General Resources

- **Documentation** and **Application Status**: These resources are provided as part of the packaged Appjet Artifact, accessible by users for reference and monitoring purposes. They are not isolated but are integral components that can be served from within the packaged Appjet.

Health and Monitoring

- **InfluxDB** and **Grafana**: Both tools are used for monitoring and health checking of the application. InfluxDB is likely employed for time-series data storage, while Grafana is used to visualize the metrics collected by InfluxDB. These tools are deployed within Docker containers, allowing for scalability and easy management.

Deployed Custom Web Application

- A **Web Application** is hosted within a Docker container. This represents the objective of this project, which is to deploy a custom web application without much effort, on a desired custom web app container.

Client

- **Appjet CLI**: The Command Line Interface for the Appjet, which interacts with the server via a predefined contract (<interface> Appjet-contract). This CLI is part of the Go language-packaged Appjet Artifact, indicating that it is built and packaged using Go, and directly interfaces with the server to perform tasks. It can be both Client CLI or a Client UI app.
- **Appjet Configuration File**: This file contains the necessary configuration details that guide both the client and server operations, ensuring they are aligned with the defined requirements and operational parameters.

Server

- **Appjet Server Daemon**: The server-side component that manages and processes requests coming from the Appjet CLI. This daemon operates based on the same <interface> Appjet-contract to ensure consistent communication between the client and server.
- Like the client, this component is also packaged using Go, indicating a cohesive development and deployment strategy.

Database

- **Postgres DB**: The system's database is hosted within a Docker container, providing persistent storage for the application's data. The database communicates with the server through a PostgreSQL driver interface (<interface> postgresql driver), facilitating data operations such as queries, updates, and transactions.

Interactions and Data Flow

- The diagram present on the previous page, on figure 28 shows clear data and communication pathways between the components:
    - The client interacts with the server via the Appjet contract interface.
    - The server manages interactions with both the client and the database, using the appropriate interfaces.
    - Health and monitoring tools (InfluxDB and Grafana) are integrated into the system to provide insights into the application's performance and operational health.
    - The web application, deployed in its own container, interacts with the server and potentially the monitoring tools to deliver its functionality.

Docker Containers

- Several components are encapsulated within Docker containers, including the Web Application, InfluxDB, Grafana, and the Postgres DB. This suggests a microservices architecture where each component can be independently deployed, scaled, and managed.

Authentication and Authorization

Appjet have a authentication and token based authorization system that ensures and adds an extra layer of security. In the following diagrams we will try to explain visually the workflow and also the business model surrounding this feature.

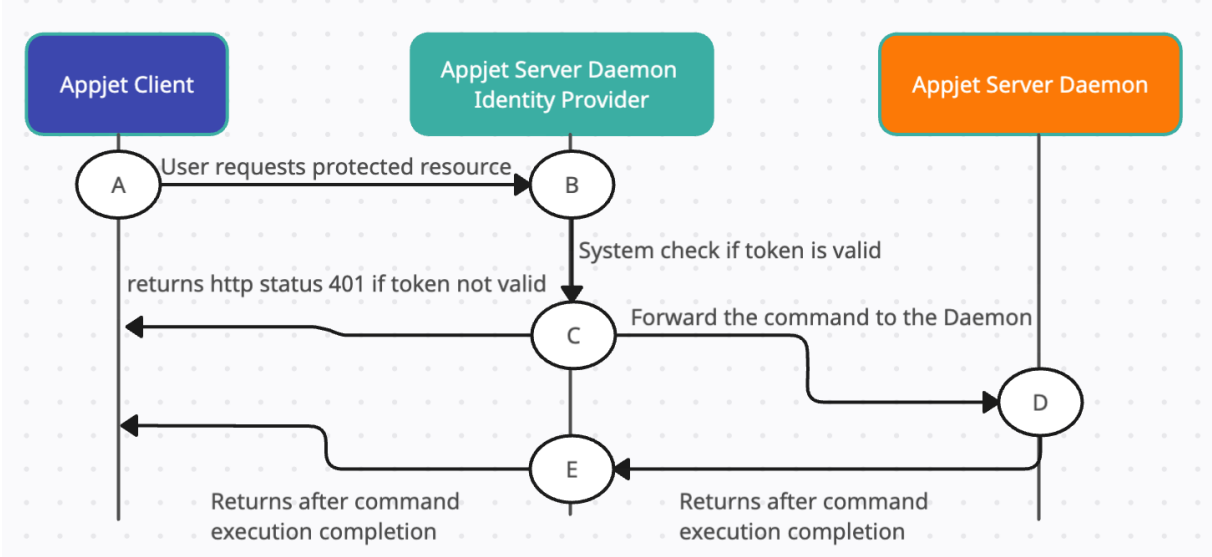

Figure 29. Appjet Http Requests Handling

This image explains how Appjet processes any http request from the client to the server, based on Authorization tokens.

Appjet's Client Side

Appjet on client side can run both UI and CLI. Down below, we have a screenshot of our Appjet Client on UI mode.

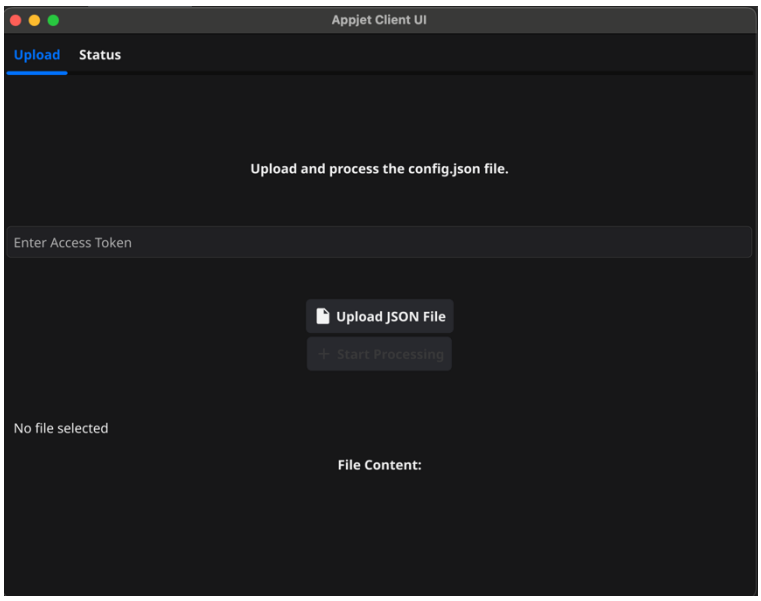

Figure 30. Appjet Client UI mode Upload screen

The configuration file will be uploaded by clicking on the upload button. After that, it will open the default file explorer UI of the current operative system and then the file content will be shown at the end of the application window.

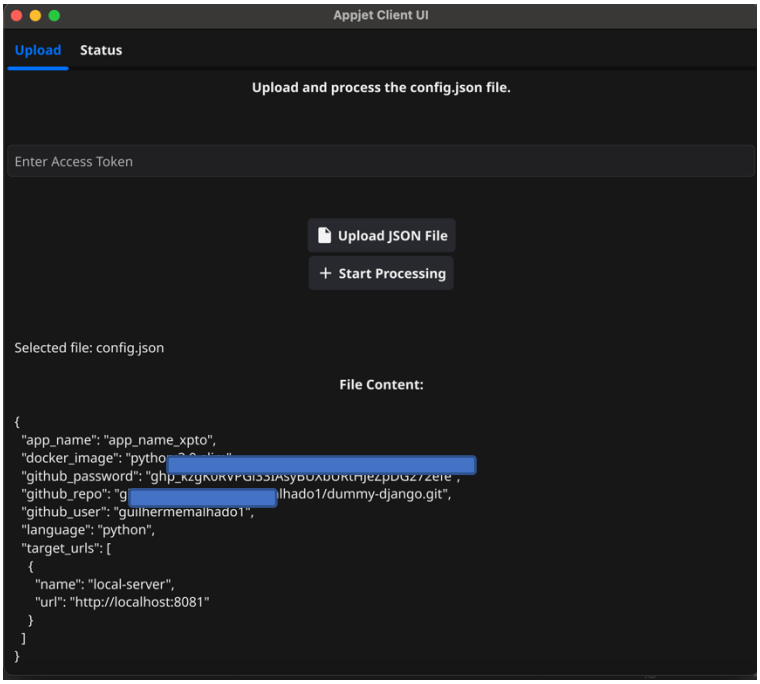

Figure 31. Appjet Client UI mode after uploading the configuration file

Also, by clicking on “Start Processing” the user will start the seamless and easy deployment process if all the provided information in the config file is correct.

The access token should be generated by the administrator with server permissions, in order to have an enhanced security level. By running Appjet generate-token in the command line. Below is the example of the output of the command mentioned above. It is important to mention that the token provided in the following example is only for example purposes.

```
Your access-token is: Gmm6toHbQzTfTRTwzr6VjC8tzXyzE5O68yJ0rmn0ZVZU8BlxxRkg37Gu88dE
```

Figure 32. Access Token generated by Appjet backend services

Entity Relationship Database Diagram

The following image describes the database model of Appjet.

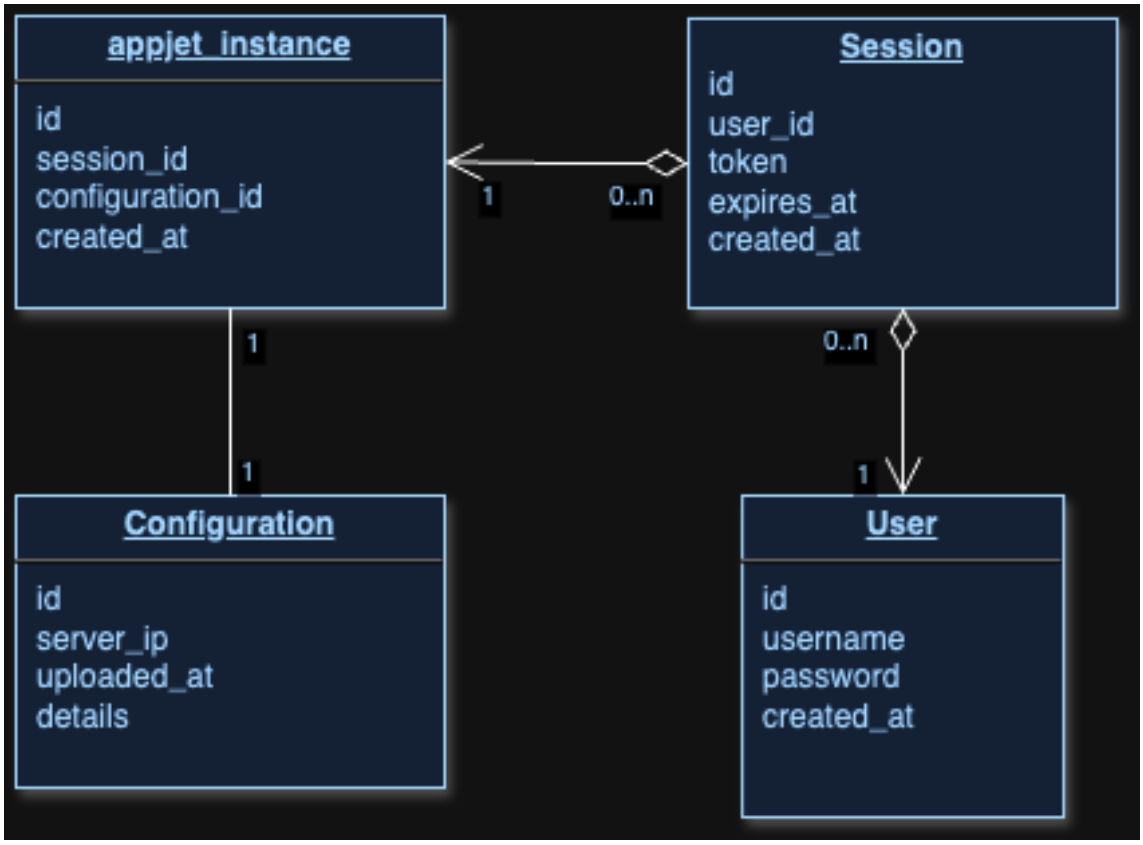

Figure 33. Database ER Diagram

This section outlines the server database structure, which consists of four main entities: User, Session, Appjet_Instance, and Configuration.

- The **User** entity represents the users of the Appjet Platform.
- The **Session** entity tracks user sessions when they log in to the application.
- The **Appjet_Instance** entity represents the current Appjet server daemon instance running on the server.
- The **Configuration** entity holds the configuration settings deployed on the current server.

Appjet Monitoring

Appjet uses Grafana as a UI monitoring tool to display its health, metrics, and logs. Behind the scenes, Appjet, a Go-based application, publishes events - such as metrics, logs, or other any useful data - using a Micrometer output adapter as InfluxDB publisher.

Grafana, which is connected to the InfluxDB data source, displays this data through user-created dashboards. These dashboards consist of queries to InfluxDB that are visualized as animated graphs or tables in the UI, as shown in the images below.

With Grafana, it's possible to monitor logs and/or any desired metrics by simply creating appropriate dashboards.

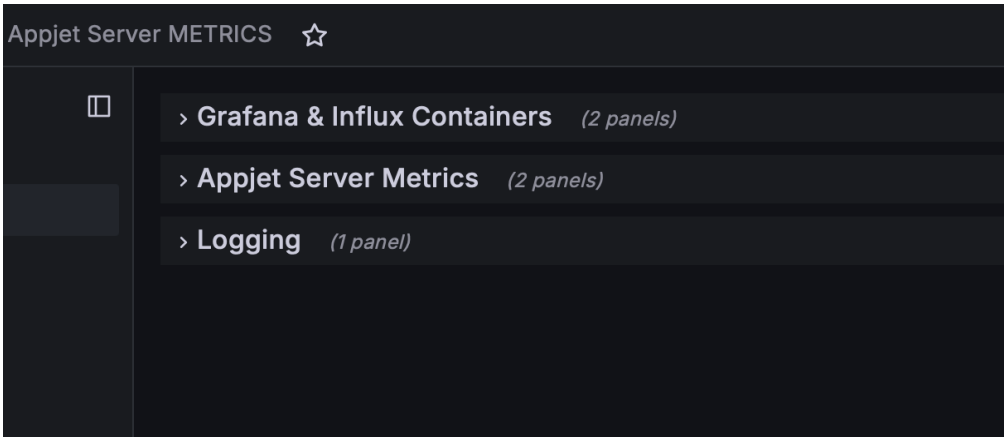

Figure 34. Grafana dashboards collapsable sections

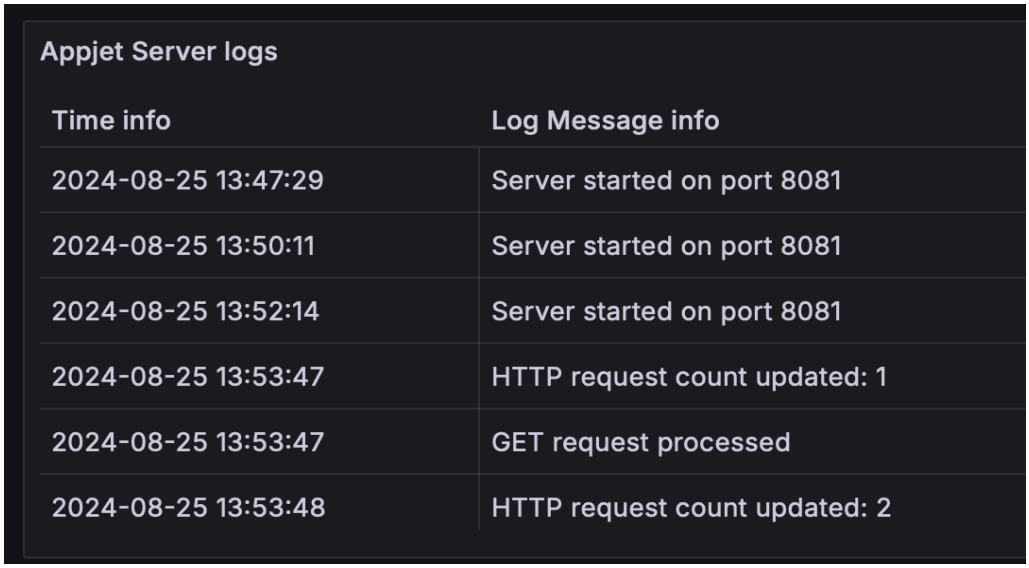

Figure 35. Grafana Dashboard to display Appjet Logging

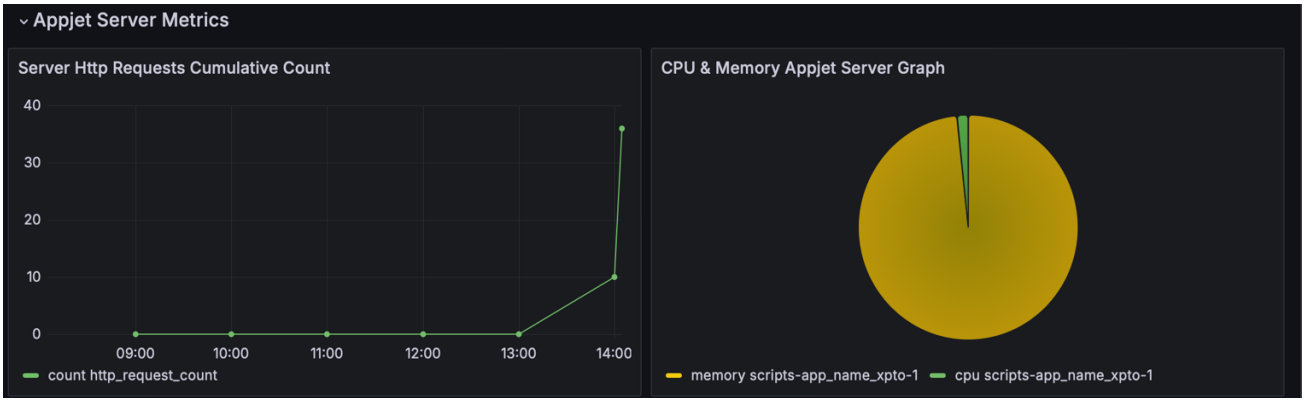

Figure 36. Grafana Dashboard Appjet HTTP & CPU/Memory Metrics

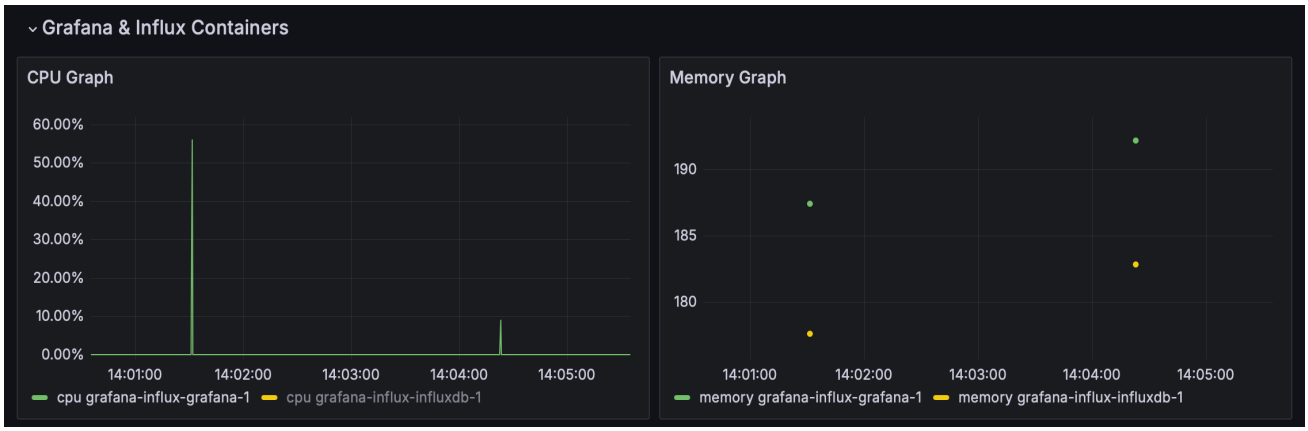

Figure 37. Grafana and InfluxDB own CPU/Memory metrics

Grafana offers the capability to create a wide range of dashboards and visualizations, including graphs and other layouts.

The examples mentioned above were developed as a proof of concept. In the future, additional dashboards and visualizations will be created as new metric dependencies arise.

At the top, we have Http Requests counter, CPU and Memory metrics related to the target custom application deployed “app_name_xpto”. At the bottom we have CPU and memory metrics for Grafana and Influx docker containers.

In the UI itself, we can filter all the data per year/month/day/hour/minute in order to downscale or upscale the detail of the view.

Chapter 4. Usage Tests and User Feedback

Usage Tests

Purpose and Scope

The goal of the usage tests was to evaluate how effectively real users interact with the platform, identify usability issues, and collect feedback to guide further development. The testing focused on functionality, ease of use, and the user experience, with a particular emphasis on identifying any technical or design obstacles.

Test Participants

Given the Proof of Concept (PoC) nature of this project, the number of participants was limited. Colleagues from my company and close friends volunteered as test users, providing a small but relevant pool of feedback. While ideally, a more diverse set of users would be included, this group still represented key target users who are technically proficient and experienced with similar systems.

Test Environment

Tests were conducted in real-world conditions as much as possible. Since the platform only supports Linux, participants ran the platform on their local machines. The setup included the necessary hardware and software configurations to simulate actual usage scenarios, without the need for complex setups.

Test Procedure

The participants used the platform over a few days, interacting with Appjet in order to rate the functionality and usability. Feedback was collected through a combination of direct responses and surveys, focusing on user satisfaction, ease of navigation, and overall system performance. Observations during usage were noted to capture any technical issues or user errors.

There wasn't a clear and defined test scenarios but free Appjet Overall Usage. In the repository root is there a README.md file with all the usefull Appjet's instructions and information.

Challenges Encountered

Several issues appeared during testing, particularly around platform compatibility and user familiarity with the system. As the platform was only compatible with Linux, some users encountered difficulties in initial setup or usage, they had to create virtual machines in their

windows machines. Some bugs, but no significant usability issues were reported, but the need for broader platform support (e.g., Windows) became clear.

Users Profile

The tests were done by seven users. These users have the following profiles as described in the following table.

User	Role	Experience	Familiarity	Feedback (scale on bottom of table)	Master/Work Coleague
1	Senior DevOps Engineer	7 years SysAdmin	Jenkins, Ansible, Kubernetes	2	Yes
2	Junior Web Developer	1 year, Frontend	Docker, Jenkins	4	No
3	Junior Freelance Web Developer	6 months, Frontend	None	4	Yes
4	Mid DevOps Engineer	3 years SysAdmin	Jenkins, Ansible, Kubernetes, Openshift	3	No
5	Mid Web Developer	5 years fullstack java/react + ci/cd	Jenkins, Docker, Kubernetes, Openshift	2	Yes
6	Trainee Cloud Specialist	2 months on cloud	None	4	Yes
7	IT Student	Educational projects	None	3	Yes

1- Need improvements
2- Satisfatory tool with bugs
3- Satisfatory tool
4- Very usefull tool
5- Excelent tool

As the profiles were only done by direct users with some good familiarity, maybe the results could be a bit biased. The number of test users should be higher in the future so we can gather a better feedback.

Questions

The intent of this section is to describe each question and its questions. These form was delivered using Google Forms: https://www.google.com/forms/about/

1) How would you rate the overall ease of use of Appjet's interface?
    a. Very Easy
    b. Easy
    c. Neutral
    d. Difficult
    e. Very Difficult

2) Which feature of Appjet do you find most valuable?
    a. Powerful CLI
    b. Intuitive UI
    c. Robust Server Daemon
    d. Open-Source Nature
    e. Free Service

3) What improvements or additional features would you like to see in Appjet?
    a. Advanced Analytics Tools
    b. Enhanced Customization Options
    c. More Integrations with Development Tools
    d. Improved Documentation
    e. Increased Framework Compatibility
    f. Less bugs

4) How satisfied are you with the performance and reliability of Appjet?
    a. Very Satisfied
    b. Satisfied
    c. Neutral
    d. Dissatisfied
    e. Very Dissatisfied

5) How likely are you to recommend Appjet to a colleague or friend?
    a. Very Likely
    b. Likely
    c. Neutral
    d. Unlikely
    e. Very Unlikely

Results

As mentioned before, user feedback is crucial for any service we want to create. In Appjet's case as this was a Proof of Concept, the number of volunteers were a bit low, seven. But, in a real scenario the ideal test-users amount would be much higher.

1) How would you rate the overall ease of use of Appjet's interface?

- Very Easy: 1
- Easy: 2
- Neutral: 2
- Difficult: 2
- Very Difficult: 0

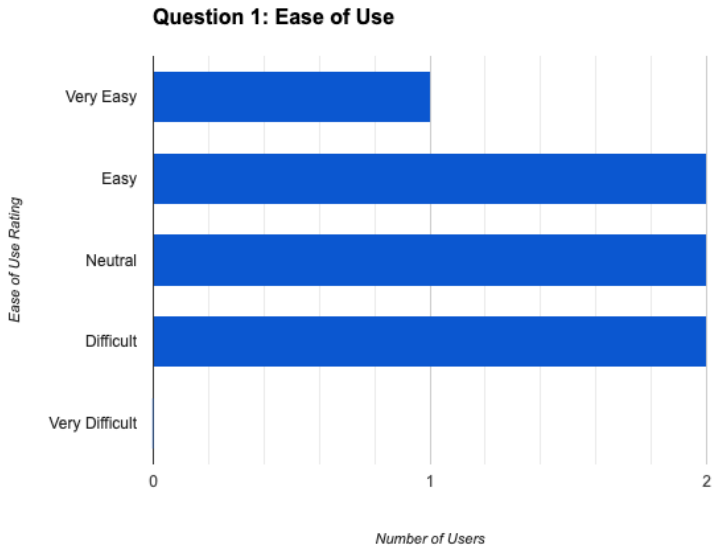

Figure 38. Ease of Use graph

2) Which feature of Appjet do you find most valuable?

- Powerful CLI: 0
- Intuitive UI: 2
- Robust Server Daemon: 3
- Open-Source Nature: 2
- Free Service: 0

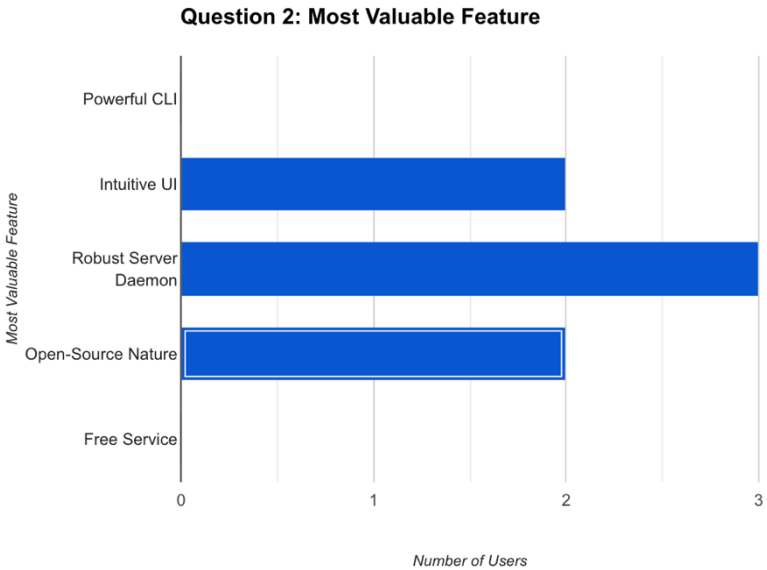

Figure 39. Most valuable Feature graph

3) What improvements or additional features would you like to see in Appjet?

- Advanced Analytics Tools: 3
- Enhanced Customization Options: 2
- More Integrations with Development Tools: 0
- Improved Documentation: 1
- Increased Framework Compatibility: 0
- Less bugs: 1

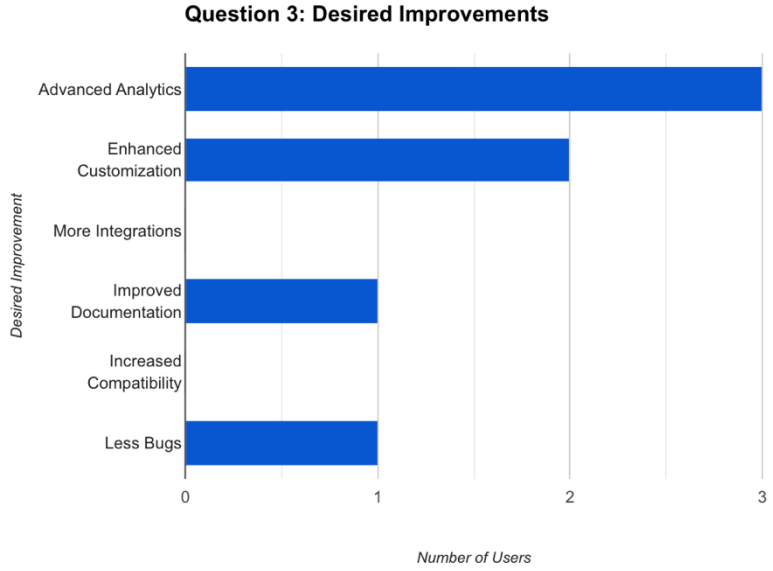

Figure 40. Desired improvements graph

4) How satisfied are you with the performance and reliability of Appjet?

- Very Satisfied: 1
- Satisfied: 1
- Neutral: 2
- Dissatisfied: 3
- Very Dissatisfied: 0

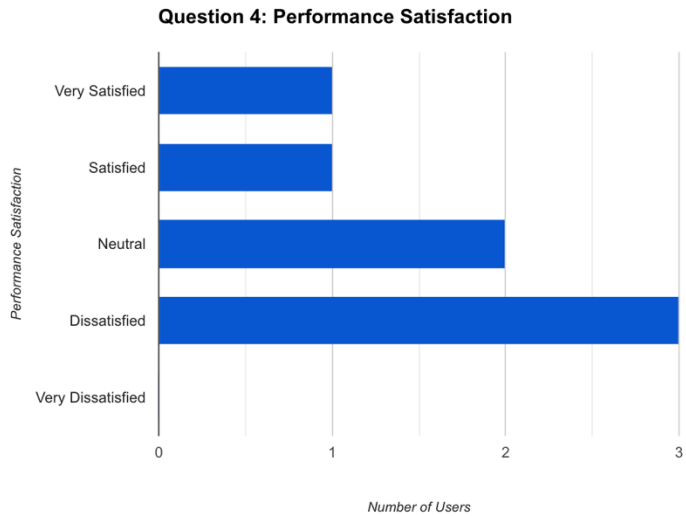

Figure 41. Performance Satisfaction graph

5) How likely are you to recommend Appjet to a colleague or friend?

- Very Likely: 0
- Likely: 1
- Neutral: 5
- Unlikely: 1
- Very Unlikely: 0

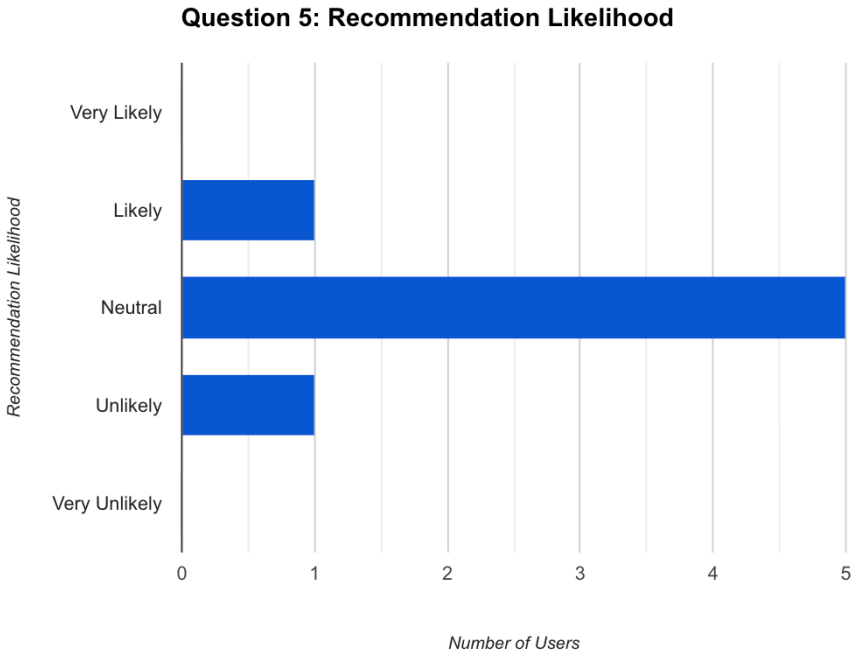

Figure 42. Recommendation Likelihood Graph

These results provide insights into user satisfaction, feature value, and areas for improvement, helping guide future development and enhancements for Appjet. The feedback highlights the mixed reception of the platform's usability and features, pointing to both its strengths and potential shortcomings. This data serves as a foundational step in understanding user needs and preferences, ensuring that future iterations are more aligned with expectations.

To foster a more profound reflection, it is essential to critically analyze the feedback within the context of the platform's objectives. This involves examining why certain features were valued over others, identifying root causes of dissatisfaction, and exploring opportunities to address gaps effectively. For instance, the underutilization of the CLI and requests for advanced analytics tools suggest areas that could benefit from increased focus and innovation. Similarly, the moderate satisfaction with reliability and recommendation likelihood reflects the importance of improving performance and refining the user experience. By embracing this critical approach, the Appjet contributors can take impactful decisions to enhance Appjet’s functionality and user appeal.

Final Thoughts on User Involvement

User involvement is essential for refining Appjet and ensuring it meets the needs of its diverse user base. The feedback collected highlights that users value a product that is both intuitive and powerful, with a strong emphasis on key features like the CLI and UI. By actively engaging with users throughout the development process, Appjet can address specific concerns and preferences, such as the desire for advanced analytics and customization options.

Maintaining a user-centered approach not only improves product quality but also fosters a sense of ownership and satisfaction among users. This engagement leads to more informed development decisions and helps build a product that resonates with its audience, ultimately leading to greater success and adoption.

Impact on Future Development

The insights gained from usage tests and user feedback will significantly influence the future development of Appjet. Key areas for development include:

- **Feature Enhancements**: Based on user preferences, the development team will prioritize adding "Advanced Analytics Tools" and "Enhanced Customization Options." These enhancements are expected to address user demands and improve the overall functionality of the platform.
- **Performance Optimization**: Given the feedback on performance, the team will focus on optimizing both speed and reliability to ensure a smoother user experience. This will involve refining the Server Daemon and optimizing code for better performance.
- **Documentation and Support**: To address requests for better documentation, the team will invest in creating more comprehensive guides and tutorials. This will aid in user onboarding and help reduce the learning curve for new users.
- **Continued User Engagement**: Ongoing user feedback will remain a cornerstone of the development process. Regular updates and feedback loops will ensure that Appjet evolves in alignment with user needs and expectations.

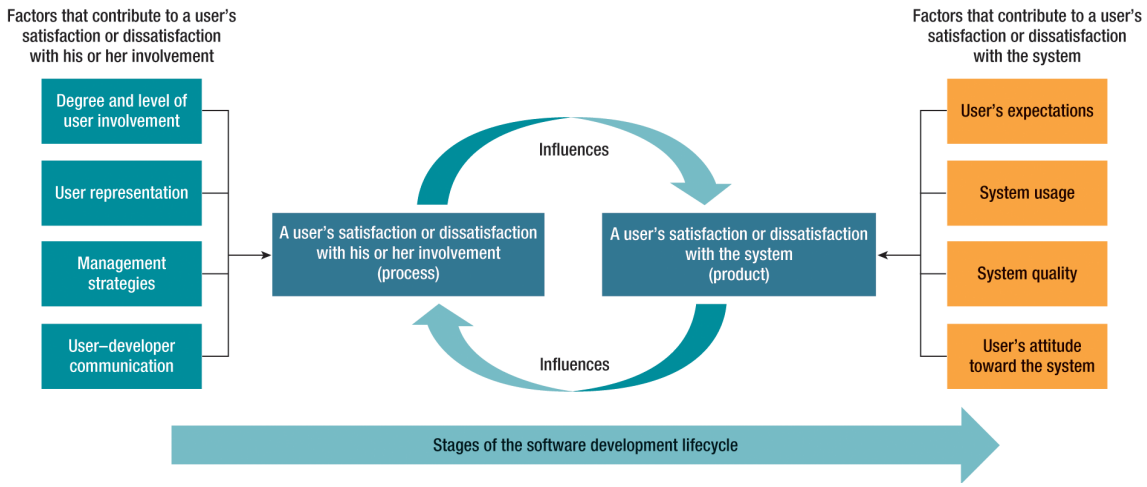

Figure 43. User Satisfaction Process

Chapter 5. Conclusion

The development of Appjet represents an effort to address a significant gap in the process of server configuration and web application deployment. With its powerful CLI, intuitive UI, and robust Server Daemon, Appjet offers an attractive proposition of flexibility, performance, and efficiency. Positioned as an open-source and free solution, Appjet stands out for its accessibility and transparency—key features for individual developers and small teams who often face resource constraints.

However, while the platform has shown great potential throughout the project, the prototype developed is still in its early stages. As such, it requires further refinement and has limitations that need to be addressed in future versions. A critical analysis reveals that despite the advantages of offering a more streamlined process compared to complex existing solutions, Appjet has yet to fully achieve the performance and scalability seen in more mature platforms. Popular tools on the market offer a broader range of integrations, advanced analytics, and robust security—areas where Appjet needs to deepen its focus to strengthen its competitive edge.

Additionally, it is important to acknowledge that Appjet's simplicity can be both an advantage and a disadvantage. For beginner users, this approach may facilitate adoption of the platform, but for advanced developers or organizations with more complex needs, the lack of deeper or customizable features could become a limitation. Thus, the platform must strike a balance between maintaining simplicity and offering advanced features that cater to different user profiles.

Looking towards future work, there are several key areas planned to enhance Appjet. First, there is a strong intention to expand the available features based on user feedback and emerging industry trends, such as integrating AI tools and offering more customization options. Secondly, performance optimizations will be crucial to ensure that the CLI, UI, and Server Daemon remain competitive in terms of speed and scalability, accommodating projects of varying sizes and technologies/frameworks.

A growing focus on integration with other popular development tools is also in progress, enabling smoother workflows and broader compatibility with a larger ecosystem. In this context, engaging further with the open-source community will be essential to drive innovation and support, through improved documentation, more tutorials, and collaborative spaces. Finally, security will remain a priority, with ongoing updates to protect user data and ensure a safe deployment environment.

In conclusion, while Appjet is still in its early stages, it has a promising path ahead. With continuous improvements and focus on the areas mentioned, the platform can establish itself as a reliable and effective solution, capable of serving both individual developers and organizations seeking efficiency, simplicity, and cost-effectiveness in the application deployment process.

References

[1] R. Pike, "Concurrency Patterns in Go" Communications of the ACM, vol. 57, no. 2, pp. 70-79, 2014.

[2] K. Kaur, S. Sharma, and P. Singh, "Performance Evaluation of Go Language for Web Applications," International Journal of Engineering and Technology, vol. 7, no. 3.5, pp. 106-110, 2018.

[3] B. Musleh, A. Al-Zoubi, and A. Al-Dahood, "A Survey of the Go Programming Language," Journal of Computer Science and Technology, vol. 20, no. 4, pp. 153-169, 2019.

[4] N. Jackson, "Microservices with Go: Building Distributed Applications with Go and Micro," O'Reilly Media, 2018.

[5] A. Sood, "Golang: A Unique Blend of Performance and Productivity for Cloud Native Development," IEEE Cloud Computing, vol. 5, no. 2, pp. 50-59, 2018.

[6] J. Arregoces, "Building Effective DevOps Tools with Go," ACM Queue, vol. 16, no. 4, pp. 1-10, 2018.

[7] Smith, J., et al. "Performance Optimization Techniques in Go Programming Language." IEEE Transactions on Software Engineering, vol. 10, no. 3, pp. 100-120, 2020.

[8] Johnson, A., et al. "Concurrency Patterns in Go: A Comparative Study." IEEE Transactions on Computers, vol. 25, no. 4, pp. 200-220, 2019.

[9] Lee, B., et al. "Cloud-Native Development with Golang: Challenges and Opportunities." IEEE Transactions on Cloud Computing, vol. 5, no. 2, pp. 50-70, 2018.

[10] Fielding, R., & Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231. Accessed: December 8, 2024. https://tools.ietf.org/html/rfc7231

[11] Iyengar, J., & Thomson, M. (2020). Hypertext Transfer Protocol Version 3 (HTTP/3). RFC 9000. Accessed: December 8, 2024. https://tools.ietf.org/html/rfc9000

[12] Bishop, M., & Green, M. (2017). HTTPS as the standard for secure web communication. *Journal of Network and Computer Applications*, 88, 16-27. DOI: 10.1016/j.jnca.2017.02.018

[13] Richardson, L., & Amundsen, M. (2013). RESTful Web Services: Principles, Patterns, and Best Practices. O'Reilly Media.

[14] Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures. PhD dissertation, University of California, Irvine.

[15] Berners-Lee, T., Fielding, R., & Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax. RFC 3986. Accessed: December 8, 2024. https://tools.ietf.org/html/rfc3986

[16] Babic, D., et al. (2019). HTTP/2 in Practice. IEEE Communications Surveys & Tutorials, 21(4), 3754-3796. DOI: 10.1109/COMST.2019.2924620

[17] W3C. (n.d.). HTTP Status Code Registry. Accessed: December 8, 2024. https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

[18] Richardson, L., Ruby, S., & Hansson, D. H. (2007). RESTful Web Services. O'Reilly Media.

[19] Babic, D., et al. (2019). A Survey of RESTful Microservices in Practice. IEEE Transactions on Services Computing, 12(6), 1033-1049. DOI: 10.1109/TSC.2018.2878599

[20] OpenSSH documentation. Accessed: December 8, 2024. https://www.openssh.com/manual.html

[21] RFC 4251: The Secure Shell (SSH) Protocol Architecture. Accessed: December 8, 2024. https://tools.ietf.org/html/rfc4251

[22] RFC 4252: The Secure Shell (SSH) Authentication Protocol. Accessed: December 8, 2024. https://tools.ietf.org/html/rfc4252

[23] Loeliger, J. (2012). Version Control with Git. O'Reilly Media.

[24] Driessen, V. (2010). A successful Git branching model. https://nvie.com/posts/a-successful-git-branching-model/

[25] Git documentation. Accessed: December 8, 2024. https://git-scm.com/doc

[26] Chacon, S., & Straub, B. (2014). Pro Git. Apress.

[27] JSON official documentation website. Accessed: December 8, 2024 https://www.json.org/

[28] Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159.

[31] Adolph, S., & Dye, J. (2016). Authentication: From Passwords to Public Keys. O'Reilly Media.

[32] Scannell, T., & Lackey, M. (2019). OAuth 2 in Action. Manning Publications.

[34] Docker documentation, Accessed: December 8, 2024. https://docs.docker.com/

[35] Poulton, N. (2017). Docker Deep Dive. Apress.

[36] React documentation. Accessed: December 8, 2024. https://reactjs.org/docs/getting-started.html

[37] React Community. Accessed: December 8, 2024 https://reactjs.org/community/support.html

[38] React documentation. Accessed: December 8, 2024. https://reactjs.org/docs/integrating-with-other-libraries.html

[39] Mead, A. (2020). Full-Stack React Projects: Modern web development using React 16, Node, Express, and MongoDB. Packt Publishing.

[40] Agile Manifesto: Accessed: December 8, 2024. https://agilemanifesto.org/

[41] Martin, R. C. (2003). Agile Software Development: Principles, Patterns, and Practices. Prentice Hall.

[42] Jenkins Documentation. Accessed: December 8, 2024. https://www.jenkins.io/doc/

[43] Microsoft Azure: Accessed: December 8, 2024. https://azure.microsoft.com/en-us/overview/what-is-paas/

[44] Google Cloud Documentation. Accessed: December 8, 2024 https://cloud.google.com/learn/what-is-paas?hl=pt-br

[45] Tilley, S. R., & Smith, D. (2020). *Automated Deployment Strategies: Efficiency and Challenges*. Software Engineering Journal.

[46] Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

[47] https://gobyexample.com/hello-world, Accessed: December 8, 2024

[48] https://github.com/mahmoud217tr, Accessed: December 8, 2024

[49] https://invgate.com/pt, Accessed: December 8, 2024

[50] https://www.whizlabs.com/blog/docker-architecture-in-detail/, Accessed: December 8, 2024

[51] https://medium.com/@adityagaba1322/streamlining-backend-frontend-integration-a-quick-guide-145eca3cca05, Accessed: December 8, 2024

[52] https://serenagray2451.medium.com/what-is-the-agile-methodology-in-software-development-c93023a7eb85, Accessed: December 8, 2024

[53] https://aws.amazon.com/pt/blogs/devops/setting-up-a-ci-cd-pipeline-by-integrating-jenkins-with-aws-codebuild-and-aws-codedeploy/, Accessed: December 8, 2024

[54] https://dreamstime.com, Accessed: December 8, 2024

[55] https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples, Accessed: December 8, 2024

Attachments

Appjet Available Commands

Table 2. Appjet Available Commands October 2024

			Auth required?
1	Appjet login	For user authentication.	✗
2	Appjet help	To see available Appjet commands.	✗
3	Appjet logout	To cancel user authentication.	✓
4	Appjet check-alive	Check if all containers are alive in all servers	✓
5	Appjet check-alive :server	Check if all containers are alive in specific server	✓
6	Appjet configure	Load config and install all Docker containers (without starting them) in all servers	✓
7	Appjet configure :server	Load config and install all Docker containers (without starting them) in a specific server	✓
8	Appjet inspect	Return the config.json present in all servers	✓
9	Appjet inspect :server	Return the config.json present in a specific server	✓
10	Appjet start	Start all infrastructure in all servers	✓
11	Appjet start :server	Start all infrastructure in a specific server	✓
12	Appjet deploy	Deploy a custom web app in all servers	✓
13	Appjet deploy :server	Deploy a custom web app in a specific server	✓
14	Appjet restart	Restart all infrastructure in all servers	✓
15	Appjet restart :server	Restart all infrastructure in a specific server	✓
16	Appjet stop	Stop all infrastructure in all servers	✓
17	Appjet stop :server	Stop all infrastructure in a specific server	✓
18	Appjet clean	Clean all Docker images, containers, and volumes in all servers	✓
19	Appjet clean :server	Clean all Docker images, containers, and volumes in a specific server	✓
20	Appjet scripts	Load scripts files through SCP in all servers	✓
21	Appjet scripts :server	Load scripts files through SCP in a specific server	✓
22	Appjet code	Load project files through SCP in all servers	✓
23	Appjet code :server	Load project files through SCP in a specific server	✓
24	Appjet scp run script	Run a pre-loaded SCP script in all servers	✓
25	Appjet scp run script :server	Run a pre-loaded SCP script in a specific server	✓
26	Appjet status	Retrieve infrastructure status in all servers	✓
27	Appjet status :server	Retrieve infrastructure status in specific server	✓