UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Infrastructură securizată pentru execuția proceselor

Andrei Tulpan

Coordonator științific:

Prof. Dr. Ing. Razvan Rughinis

BUCUREȘTI
2024

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DIPLOMA PROJECT

Secure infrastructure for process execution

Andrei Tulpan

Thesis advisor:

Prof. Dr. Ing. Razvan Rughinis

BUCHAREST
2024

CONTENTS

SINOPSIS

În contextul tehnologic actual, dinamic si in continua evolutie, produsele software trebuie să fie rapide, scalabile, adaptabile și mai presus de toate, sigure. Această teză prezintă implementarea unui utilitar modern, bazat pe trei principii fundamentale ale ingineriei software: securitate, eficiență și scalabilitate. Produsul este potrivit pentru mai multe categorii de clienți: atât pentru companii, cât și pentru utilizatorii individuali, care au nevoie să-și automatizeze mediile de rulare ale diferitelor aplicații utilizate, într-un mod sigur, pe dispoztive situate la distanță, menținând în același timp un control complet asupra datelor transferate. Acesta stabilește o infrastructura gazduita local, similară unui cloud privat, dar gestionat integral de client, fără dependență de alte părți terțe. Produsul are două tipuri principale de clienți: pe de o parte, companiile care trebuie să respecte cu strictețe standardele de securitate, cum ar fi cele din domeniile financiar, guvernamental sau medical, iar pe de altă parte, sunt persoanele pasionate de tehnologie, care pot utiliza produsul în scopuri proprii. Proiectul se concentrează pe automatizarea mediilor de rulare pentru aplicații software, având ca obiectiv dezvoltarea unei soluții care să gestioneze eficient aceste sisteme cu o intervenție umană minimă. Acest proiect reduce posibilitatea apariției erorilor, îmbunătățește administrarea și accelerează implementarea prin utilizarea tehnologiilor moderne și a tehnicilor de automatizare. Lucrarea de cercetare urmărește în detaliu implementarea produsului si modul in care s-a ajuns in stadiul curent, evidențiind atat beneficiile și îmbunătățirile aduse în diferite scenarii de utilizare.

ABSTRACT

In today's dynamic and evolving technological context, software tools must be fast, scalable, adaptable, and above all, secure. This thesis presents the implementation of a modern tool grounded in three core principles of software engineering: security, efficiency, and scalability. The tool is suitable for both companies and individual users looking to securely automate application runtimes in a remote environment while maintaining full control over their data. It creates a locally hosted environment similar to a private cloud that is entirely client-managed without third-party dependencies. Customers are divided into two categories: enterprises requiring stringent security compliance, such as those in finance, government, and healthcare, and technology enthusiasts. The project focuses on automating runtime environments for software applications, aiming to develop a solution that efficiently manages these systems with minimal human involvement. This reduces the possibility of errors, enhances administration, and speeds up deployment by implementing modern technologies and automation techniques. The research thoroughly examines the tool's implementation, highlighting its visible benefits and improvements in various scenarios.

ACKNOWLEDGMENTS

I would like to thank first to my thesis advisor, Razvan Rughinis, who supported me from the beginning with my personal project idea and always encouraged me to present it at various conferences. Without his help, it would have been a much harder journey.

And last but not least, special thanks to my family who supported me all these years. Thank you for all the support you have given me. It meant everything to me and helped me substantially in reaching this stage in my life.

1 INTRODUCTION

In this thesis is followed the implementation stages of a modern software tool, based on the most important three principles of software engineering: security, efficiency and scalability.

Our product it's a perfect match for all the clients which need an automation software for their applications runtime, using a secure environment running on multiple physical nodes, with a complete control over the data in use.

To achieve these the tool is creating a hosted on-premise environment acting like a private cloud, fully managed by the client without third-party intervention or dependency.

The customers of our product are divided in two main groups, the first category is represented by the companies and the second one by individuals. The businesses are the one which require enterprise conformity to the existing security standards, such as the finance, government or healthcare. On the other hand, the other category is represented by tech enthusiasts which need a plus of power in their life.

1.1 Context

Currently times after the pandemic of from 2020, the dependence of modern devices connected to the internet has increased significantly. According to the scientific publication by Luca Cerniglia, Silvia Cimino, Eleonora Marzilli, and Giulia Ballarotto, the number of individuals utilizing the internet for both recreational purposes and professional or educational activities has shown a twofold increase in the 4-6 hours per day category, while the 6+ hours per day category has seen an almost fourfold rise.[2] In 2023, 67% of world population was part of the online network and this number is continuously growing from year after year, in 2005 were approximately 1 billion individuals online and now are roughly 5.4 billions based on the statistics between 2005 and 2023 made by Ani Petrosyan. [15]

Considering the significant increase in online consumers, every company must line up with this new digital world and should include their services there. For example, many restaurants joined to a catering application or developed a personalised and abandoning the old approach with operators which present the offers and take your order, the consulting firms moved their activity completely in online saving money from the space rental and also saving clients time to travel to the their office, medicine appointments that do not necessitate in-person examinations are moved in online, shopping markets had a big increase in revenue from online orders with door delivery and much more scenarios like these, which created a lot of new job opportunities for work-from-home.

Once with the expansion of online services, all businesses need to join on this new market and need services to automatically manage infrastructure for their services. Because of this desperately need, they request help to public cloud providers, such as Amazon Web Services[1], Microsoft Azure[5], Oracle Cloud[6] and much more others. The market for cloud computing services has grown substantially, from $145 billion in 2017 to $675 billion currently, and is projected to reach $824 billion by next year, according to an analysis by Lionel Sujay Vailshery[25].

Furthermore, the lately usage growth of artificial intelligence (AI) and machine learning (ML) technologies, resulted in an increase adoption of cloud-based computing. This new trend not only increased the dependence of the companies to cloud services, but also increased efficiently the innovation in multiple areas, from healthcare to finance, retail and entertainment. Due to the rapid expansion of the artificial intelligence domain and cloud services dependency, it made a significant contribution to the previously mentioned market increase of these cloud services. The revenue of the artificial intelligence market increased from $13 billion in 2020 to almost $900 billions in 2023 and is expected to reach $1.3 trillion in 2032 based on the statistical date made by Bergur Thormundsson[23].

1.2 Problem

The usage of cloud platforms comes with a list of some concerns. First of them and the crucial ones are the security issues, because on cloud based systems your entire data is stored in a environment which is beyond your control, furthermore all activities must travel through the global area network from your computer to the cloud based physical server location, making them vulnerable to numerous cybersecurity attacks.

Using of cloud providers by businesses can bring additional legal concerns, because some countries have rigorous laws related to the users data storage and most providers store their data in regional centers, which can be located in other countries. Others concerns can come from the cloud providers who have transparency issues, such as data ownership, geographic locations of data storage, third-party access to data, data handling and security practices.

Network dependency is another topic that must be taken into account, because in case of real-time applications, the distance between the cloud, producers and consumers introduce a latency which affect the performance. Another issues which can rise here is the limited bandwidth provided by the internet service provider, in this case the data transfer between the business endpoints and the cloud service can be limited at low speeds.

Most of the cloud providers follow the payment type of pay-as-you-go model, which can lead the clients to unexpected costs, if they don't monitor close the resource usages. Furthermore, other unexpected costs can come from both the internet service provider and cloud provider, in case of huge blocks of data are moved between cloud physical location and client's machines.

1.3 Objectives

The concept behind this project is to bring the power of a cloud computing infrastructure into a self-hosted environment, where benefits from the both world join to create a new powerful system made to meet the needs of consumers.

The idea of this project it's a perfectly match for a wide range of consumers, from the ones that needs huge computational power, such as big enterprises, to small businesses, or even individual developers, who need environments for testing, automation of CI/CD pipelines or just automation of daily necessities. Our clients have a lot of benefits from infrastructure and environment customization and self-management to high adaptability and versatility.

The main objective of this presented idea is to satisfy user needs and to provide the best experience, without the bottlenecks and the costs that can appear in case of the cloud providers. In addition, the customers control the entire infrastructure and have complete authority over the data storage, solving the related problems with privacy and security.

Our product need to ensure wide compatibility between any environments and any devices, from embedded devices to super computers and from UNIX-like operating systems, such as Linux and macOS to Windows machines. In this way, we provide flexibility for the users to choose both the best hardware components and software operating system for their needs. Also users can have an infrastructure composed from multiple machines with different operating systems, bringing interoperability between them.

Our main advantage is that the data transfer it's not limited by external factors, such as internet service provider (ISP), intermediary routers or cloud service infrastructure and it's limited only by the local area network (LAN) infrastructure, where all the infrastructure's components (the server and the compute nodes), are running.

1.4 Solution

To achieve the best optimization and high compatibility, we chosen C++[4] as main programming language. This decision was made following multiple selection criteria, such as: low-level memory access, low overhead, greater community, a lot of libraries and embedded devices compatibility. By making this choosing, our results will be highly optimized and efficient and comes with possibility to fine-tune the performance, depending the scenario where they are used. Furthermore, our components are working with a large area of devices, from low power, such as embedded devices, to very high power, such as super computers.

The essentials components of our project are the server and the nodes, the communication between them is made over an encrypted ethernet transmission. The server is the manager of the entire infrastructure, which connect and control all the nodes. Also the server is responsible with the external communication with web control panel. The other mentioned component, the node is the endpoint where the applications are executed.

The communication system between the server and the nodes is made on top of sockets and is encrypted end-to-end with an ephemeral key generated with a key exchange algorithm, preventing attacks of type "eavesdropping". Even with a process of disassembly and decompilation made on the executable it's impossible to find any clue to recreate the encryption and decryption key.

The main server accept a predefined type of requests, over an application programming interface (API), provided to the web control panel using a websocket. This part of the communication is also secured and encrypted, using Transport Layer Security[11] (TLS) on top of the communication socket.

One of the most required componenets is the live metrics one, that is responsible to collect the metrics from the node and to send them in real-time to the consumer. Another component which improve the reliability of the system is the fault tolerance mechanism, which is supervising the execution of the tasks, and restarts them in case of a node fault.

1.5 Results

The project managed to implement the chosen solution and to reach the previously presented objectives and now I will discuss about some testing scenarios that I performed to check the system's performance, security, and reliability under various conditions.

First test was represented by execution of multiple processes and measure the real-time output, which wasn't affected by the fact that data was sent over network and not directly shown in the console.

The next scenario was represented by multiple processes which use enormous computational power and a process which use intensive I/O operations to measure it's response time, which was again the same as it was shown local, not in another machine.

The security system was tested by sending a lot of flood, using garbage packets, in the context of trying to replicate a Denial-of-Service (DoS) attack. The results of this testing scenario were positive, the system blocked automatically all the garbage connections and maintained it's stability.

Another security attack the system was subjected to was a penetration one, more precisely a Man-in-the-Middle (MitM) attack, where the packets were intercepted in order to obtain the encryption key and to decipher the messages, the system was impenetrable and passed the scenario without issues.

1.6 Thesis structure

The next chapters will explain in more details every concept presented in the introduction, going with further explications about the technical and business ideas, and focusing on providing a better understanding of the project main idea. Following this chapter, will be six more chapters, every of them with a specific primary idea to explain.

Chapter 2 presents the entire project with all of his functionalities and the motivation and origin of the idea to develop this project.

Chapter 3 explain the main differences between our product idea and the market existing products, focusing on our improvements and on the parts that we should integrate in our product, being "State of the Art" for this product.

Chapter 4 describe more broadly the chosen solution, explaining the foundation and presenting architecture of the implementation.

Chapter 5 represents an ample view about the implementation methods, the used libraries and other tools that helped in the process of creation.

Chapter 6 represents the analysis of the final implementation of the project and dive also in the testing scenario which validate the final thoughts.

Chapter 7, being the last one provide a point of view over all the chapters, starting with the initial objectives which was integrated in our product, following with the implementation of them and going further to the results obtained with our solution. All of these concluding with potential improvements which should be added in the future.

2 MOTIVATION

The idea behind this project was initially different, but after a presentation of this unpolished idea at a Start-ups Hackathon in Bucharest and some discussion with a few possible investors, we refined the project into its current form. The interactions with the investors and mentors introduced us in the world of potential clients and provided essential feedback, which helped us a lot.

2.1 Initial concept

The initial idea, was a tool that helps especially the freelancers to deliver their small applications, such as scripts for a certain job, as subscriptions, without the need to implement an API for that. The frontend idea of the project was a marketplace for developers, but the backend part was almost the same with the one implemented in the current idea of project, not as refined like it is now, but with the same functionalities. The idea of this platform, was focused on freelancers, who can add their products on it and also some computational nodes (self-hosted by them), where the products should be run, by their clients.

As presented above, the initially project idea was bigger and a little different, but after the presentation at the Start-ups Hackathon in Bucharest, we changed a bit the idea and we moved to another type of clients. This changes made to the idea, were conducted by the feedback received from the potential investors, who helped as to polish our idea into the one presented in this thesis.

2.2 Feedback

After the discussions with the mentors and potential investors at the mentioned competition, we received a lot of suggestions to enter to the enterprise market, because our functionalities are very sought in companies. Following this collected feedback, we redefined our idea into a tool which can manage the process execution through a web panel. Regardless the fact that we focus mainly on the enterprise market, we still want to offer some of the functionalities to the other category of possible clients.

Following the collected feedback, we created a list with the main key points of our software, that will define our infrastructure:

- Security: end to end encryption with ephemeral key
- Flexibility: adaptable infrastructure to new technology
- Control: full control over the infrastructure and the data in use
- Compliance: meet all the enterprise requirements
- Cost: optimized cost without compromising other components
- Availability: implement redundancy, failover mechanisms and recovery plans

2.3 Key features of minimum viable product

After a detailed analysis made on the plans created, we established a well defined minimum viable product (MVP) component for both our projects mainly parts: the web control panel and the secure infrastructure for process execution, the second one being the subject of this thesis.

The web application main responsibility is to handle the interactions between the users and the infrastructure, through the backend server., The control panel should have a simple and intuitive user-interface and multiple features, such as:

- Users Types: every defined infrastructure has 2 types of users: administrators and users
- Access Control: administrators can set permissions for users and manage their accounts
- Node Registration: administrators can manage and add nodes to the infrastructure
- Automated Backups: system can automatically back-up data at regular intervals to prevent data loss
- Dashboards: users can customize their dashboard to display the most relevant information
- Alerts: system can send notifications to users about system performance or security concerns
- Resources: users can manage resources such as storage, bandwidth, and compute power according to their needs
- Reporting: reports can be generated on usage statistics, performance metrics, and other important data to clients
- Scheduling: tasks can be scheduled by users to performed automatically at specific times
- Services: provides a list of available services that users can request
- Activity Logs: tracks user activities for auditing and security purposes

The deployed infrastructure is managed by the main server, being a part of it but also the manager and the point of control from outside. It is responsible with multiple features, such as:

- New Services: it quickly deploy new applications in the infrastructure
- Updates: manage the software updates and patches to the infrastructure
- Firewall: detects automatically network attacks and block them
- Security: implement the newest security policies
- Execution: process tasks and send them to the corresponding nodes
- Live Outputs: transmit live the streams for output and errors
- Live Metrics: transmit live metrics about the system

2.4 Implementation and architecture

The implementation will be explained and divided in small componenets in the next chapters, to gather a better understanding over the implementation design. On of the best parts of this implementation is the adaptiveness of it, matching on all the cases discussed with the potential clients, ensuring compliance with all the requirements. To test the current status of the product we choose to use real-world scenarios to present how the tool will act in productions, managing complex infrastructures and integrating multiple nodes to seamlessly work together. Additionally, we will discuss about the received feedback from early adopters and how it helped us to improved our solutions.

The architecture of the solution is based mainly based on scalability and flexibility to make the product one of the best tools. Every component of it will be detailed and it will be presented the interactions between them, through this steps we aim to provide a comprehensive understanding, exploring how the system incorporates all the clients requirements in an efficient way.

3 STATE OF THE ART

In this chapter, will be discussed the advantages and disadvantages of the most advanced solutions from the market, understanding how they work and what we can improve here. Current research on the market helps us fill the gaps in current available products and to bring about a better solution.

One of the best solutions from the actual market are Ansible[18], SaltStack[22], Puppet[21], and Chef[20], but in the last few years an indirect type of competitor appeared, the cloud platforms. These mentioned platforms comes with a lot of features, but also with a lot of disadvantages, being a lot of potential for improvements. Some of example of this type of platforms are: Amazon Lambdas from Amazon Web Services[1] (AWS), Azure Functions from Microsoft Azure[5], and Cloud Functions from Google Cloud Platform[3] (GCP).

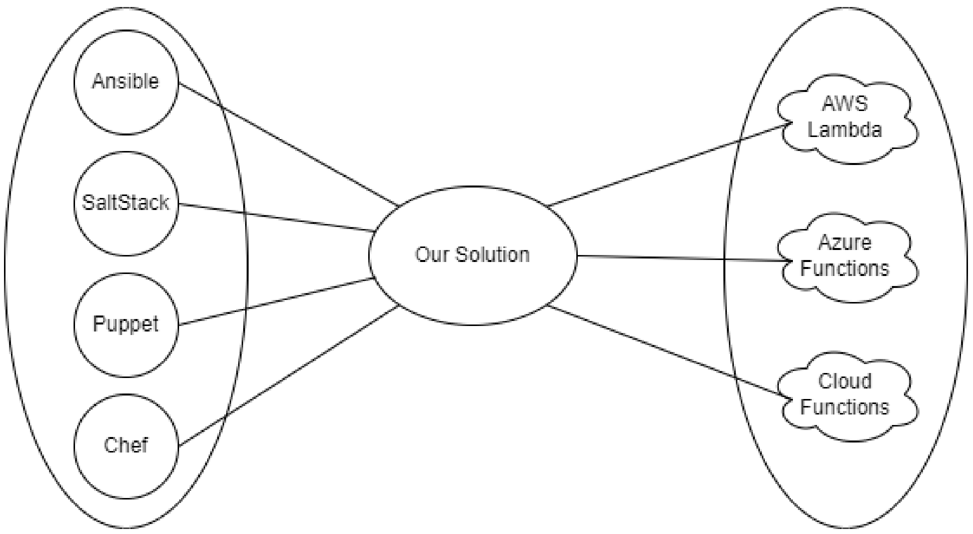


Figure 1: Direct and indirect competitors

3.1 Direct Competitors

These competitors are represented by similar platforms, which can be used on clients owned machines, with full control over the environment, or even on a cloud platform, but in a virtual machine (VM), where you can maintain the control, not in environments like the one presented as indirect competitors.

3.1.1 Ansible

Ansible[18] is an open-source tool for tasks automation, application deployment and configuration management, similar to our tool, the main difference is that is using an agentless architecture. Due to this approach, the platform has some drawbacks:

- **YAML Playbooks**: It uses playbooks for the config files, based on YAML, which is hard to use for complex logic. We are using a intuitive designed user interface, which makes the process of config very easy and intuitive.
- **Communication System**: Because it's an agentless architecture, it is using SSH as the communication system, which need to open a port to the wide area network, requiring modifications on the router side. Our solutions is based on the policy of "no ports forwarding" on nodes.
- **Real-time execution**: Relying on SSH for communication, the real-time output can be affected in comparison with the infrastructures with continuously running agents. Our infrastructure is using it's own communication system, which have an increased speed, with the same level of encryption.
- **Live Metrics**: It doesn't have a built-in solution to deliver live metrics and it needs to use a plugin or an external tool. In our solution, the live metrics are integrated into nodes and user interface.
- **Dependency Management**: Dealing with dependency between tasks, it's challenging and needs a lot of configurations through YAML files.
- **Performance**: In case of large-scale infrastructures, it could have performance issues on complex operations. Our tool is working on every device from embedded devices to supercomputers due to C++[4] magic, instead of Python.
- **Scalability**: Large deployments may require using of additional tools and adjustments to work as expected. Our solution it's working without any external tools, no matter how large is the deployment.

In conclusion, Ansible[18] can be an option, but it has a lot of drawbacks compared to our platform. The single advantage of it is the big community, which develop plugins and playbooks, but a good platform will bring powerful communities too.

3.1.2 SaltStack

SaltStack[22] is also a open-source tool for orchestration and configuration management, similar to our tool, using the same paradigm as us, with an agent on every node. Even it's more similar to our product, it still has some drawbacks:

- **Complex Set-up**: Is also based on YAML, like the previous one and is very complex to set-up and configure. Our tool has a well designed user interface which makes the configuration process easy and intuitive.
- **Communication System**: The communication system is based on ZeroMQ, which can be a good alternative to our real-time broker. Our infrastructure communication is based on an own system based TCP[10] connection and on Kafka[19] for real-time data to the user interface.
- **Live Metrics**: Similar to Ansible[18], it doesn't have an integrated system to provide live metrics, but it can be implemented using external tools. As previous mentioned, our tool has an integrated system for live metrics.
- **Dependency Management**: Managing dependencies are very problematic, especially when upgrading SaltStack[22] version.
- **Inconsistent Quality**: Some modules provided by SaltStack[22] are less stable or well-documented than others. We are using only essential modules, which are very easy and intuitive to use.
- **Performance**: Issues related with performance can appear in large environments, especially with many clients (minions), and performance can be affected by high memory and CPU usage. These are resolved in our tool, using C++[4] and it's magic, instead of Python.
- **Scalability**: Large scale deployments can be challenging, and are multiple reports where after a number of minions show up some difficulties with the infrastructure.
- **Community**: Even with a large community, it's not the active one, having problems with support responsiveness.

In conclusion, this tool has some improvements, such as the real-time messages, agent on nodes, but it came also with drawbacks like performance, scalability and also a weaker community.

3.1.3 Puppet

Puppet[21] is another open-source configuration management tool that helps to automate the management, configuration and deployment of applications on the created infrastructure, such our tool. Even it's very similar as idea with our tool, it has drawbacks, such as:

- **Complex Set-up**: It is very hard to config, without experience in tools configuration and need a lot of time to read the huge documentation, due to the domain-specific language (DSL) for the config files.
- **Communication System**: The communication system is based on http calls, which make the communication very slow, compared with our system based on bare-bone TCP[10] , mixed with Apache Kafka[19] for real-time communication for user interface.
- **Live Metrics**: They are 99% not existing, by default at every 30 minutes it's send a request over http to get the node status. Our system transmit real-time live metrics and can send from the time scale of milliseconds.
- **Real-time Changes**: The configuration changes through the nodes may be not applied immediately, and it depends on the agent's run interval. This can be a big problem for critical updates, which in our case are send over a TCP[10] socket.
- **Dependency Management**: Dependency handling between different resources can be very complex and it needs configurations hard to maintain.
- **Performance**: It's a tool resource-intensive, due to the language, Ruby, which is very easy to use, but at the same time it's not suited for performance-critical applications.
- **Scalability**: For normal environments it's scale well, but for large environments, it needs significant maintenance and attention for the infrastructure.
- **Community**: Even with a strong community, it can't provide a diversity of modules and also can't maintain up-to-date the existing modules.

In conclusion, this tool has an extremely poor performance in both the management system and the network communication system. It misses a lot of functionalities, such as real-time critical changes, and even with a big community, it can handle up-to-date modules.

3.1.4 Chef

Chef[20] is an open-source automation platform, enabling to configure and maintain the infrastructure through the code, turning the entire infrastructure into a code. Even with these new implementations of infrastructure design, it has a lot of drawbacks.

- **Complex Set-up**: The set-up of the environment is based on recipes and cookbooks, which are using Ruby-based DSL. This can be very challenging for users that aren't familiar with the programming language Ruby, which it's not even a top 5 language.
- **Communication System**: Similar to the previous tool, the communication system is based on http requests, which result in a slow communication compared with our systems made for real-time communication.
- **Live Metrics**: Live Metrics System is built-in, which is a advantage, compared to the previous presented tools, but the live system is based on http requests, which are not the best solution for real-time data.
- **Real-time Changes**: Like in the case of Puppet[21], the client runs periodically, by default at every 30 minutes, which means that critical changes can be missing until an update request is made to the main server.
- **Dependency Management**: Managing dependencies between cookbooks and recipes can become very complex, leading to difficult configurations and management.
- **Performance**: It's not the most efficient tool, being developed in Ruby, similar to the previous one, it may require a lot of resources to run at high-throughput.
- **Scalability**: It can handle small deployments, but the large one can be challenging and requires significant maintenance to ensure efficient results at scale.
- **Community**: Even with a solid community, and a lot of cookbooks, some users find the quality and the maintenance of these inconsistent.

In conclusion, Chef[20] is a tool with a poor performance and a real-time system not too well designed for a high-throughput data transfer, it is hard to configure and to learn how to do it, and even with a big community, it cannot handle up-to-date and quality cookbooks.

3.2 Indirect Competitors

This category of competitors is represented by products from cloud providers, such as: AWS Lambda, Azure Functions and Cloud Functions. These have a lot of advantages, but these are coming with a cost which need to be paid, such as:

- **Scalability**: The infrastructure doesn't need to be managed manually, it scales with the number of operations, but at the same time you don't have any control over the infrastructure where your code is executed.
- **Cost Efficiency**: It can be very cheap for new projects, but it can increase rapidly when you have a lot of traffic.
- **Event-Driven Execution**: It can trigger events using http requests, which introduce a lot of overheard, comparative with our system over a websocket.
- **Language Support**: They are limited with the language support, in our case any language can be configured, depending on the host OS.
- **User Interface**: This one is very complex and hard to use, and we want to bring a new simplified interface made from any type of users from newbie to infrastructure administrators.

In conclusion, we want to bring the best ideas from this competitors into our own idea, which is using classical model with an infrastructure running on own devices, or even in cloud virtual machines (VMs), but with a full control over it.

3.3 Our Solution

Our solution offer the best communication system between all of the present products, implementing a very fast communication for requests on TCP[10] sockets inside the infrastructure, and for the outside communication with the control panel, it use an websocket for the requests and Apache Kafka[19] for real-time data.

For the purpose of security, we are using the best library that we found, Mbed TLS[24], which provide all the necessary functions from safe key exchange, to symmetric message encryption and certificates. Also it is very efficiently, and can run on any device from super-computers to embedded devices.

The programming language, compared with the ones used by our competitors, gave us a boost of efficiency and performance, using all the power from the machine in an efficient way.

4 SOLUTION

As presented in the introduction, the proposed solution implement two main components: a server which acts as an orchestrator for the generated infrastructure and some computational nodes, which make up the infrastructure.

This infrastructure deserve to a large system of remote process execution on nodes and to the possibility of automation of these executions of the desired processes. All of these functionalities were implemented with focus on security, efficiency, scalability and reliability, to deliver the best experience for all the customers, unconcerned about the category of them.

The solution is suitable for any type of customers from large corporations, which are using servers with heavy computational power, to small businesses, freelancers and tech enthusiasts which are using regular servers, personal computers or even single-board computers (SBCs) with a limited computational power.

To be able to achieve all the objectives, I choose to develop both components in C++17[4] due of its support for low-level memory access, high performance results, fine-grained control and cross-platform development. Along with these highly valuable features it comes also with high-level paradigms, such as templates, object-oriented paradigm and interoperability, which help to a faster development.

For a wide range of compatible devices, who choose to use only essential and necessary libraries, we choose the best one for us, classifying them based on efficiency and excellence of the results. By following this approach, our components can be executed on any device from small embedded devices to powerful supercomputers.

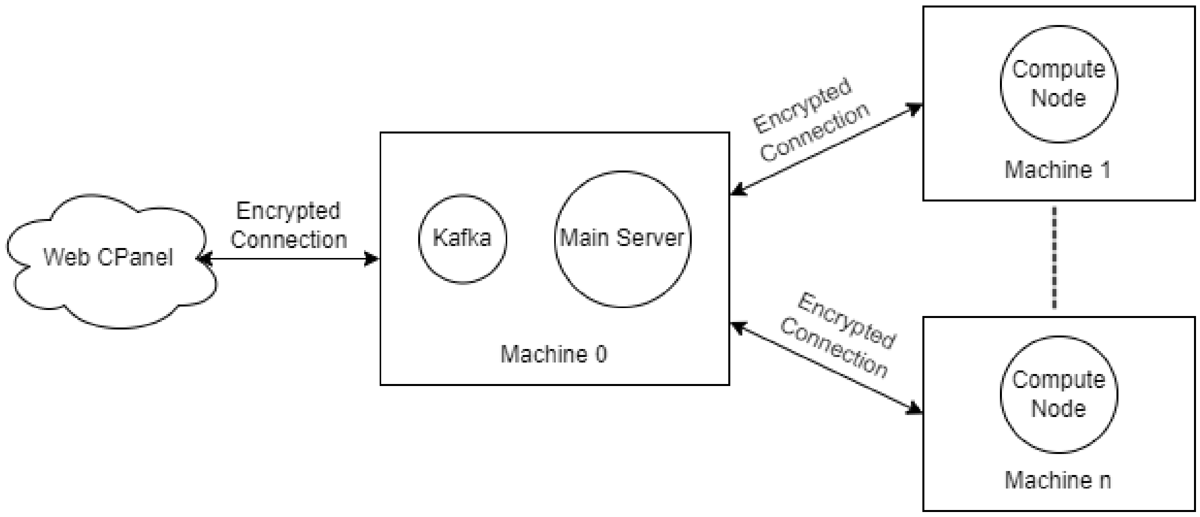


Figure 2: High Level Topology

As illustrated in the high-level topology diagram (Figure 2), all the components of the whole project are presented, but we will discuss the ones that are the subject of this thesis: the main server, called internally in the project "nexus" and the compute node, called internally "entity".

The structure of this project is divided into three sections, every of them with it's own components, as shown in Figure 2: the common part, the exclusive nexus part, and the exclusive entity part. In the following subsections, these components will be discussed in more detail.

```
├── Nexus
│   ├── QuantumNexusLink
│   ├── QuantumExchange
│   ├── SynapticCache
│   └── NetGuard
├── Entity
│   ├── QuantumCalculia
│   ├── InsightPulse
│   └── KafkaWrapper
└── Common
    ├── ParagonEnigma
    ├── QuantumInspector
    ├── QuantumSentinel
    ├── SynchronizedThread
    └── Utilities
```

Figure 3: Project Structure

4.1 Nexus Component

The main server, often referred to Nexus in the project framework, is the key component of the project, being responsible with communication outside the infrastructure, through QuantumExchange component, with communication inside the infrastructure and planning of the compute nodes, through QuantumNexusLink component. Besides these, it uses a component called NetGuard to protect against Distributed Denial-of-Service (DDoS) attacks and a component called SynapticCache, which increase the reliability of the system, minimising consequences in case of a node failure.

The communication implemented between nexus and entities it's not following the classical model of client-server model, where clients make requests and the server responds to them. In our case, we implemented an important feature, built on top of a TCP[10] server, where all the nodes initialize the connection, then step into the authentication process and then into the open communication process, both of them being controlled by the QuantumNexusLink. Through the communication process, the server receives requests from the QuantumExchange component, processes them, and then sends them further to the corresponding client, which responds back to the server.

4.1.1 QuantumNexusLink Sub-Component

On every connect request received over the TCP[10] socket, which pass through the NetGuard filters, will be created a new instance of this component with a dedicated thread, which will respect a well designed protocol to authenticate the node and after to initialize the communication.

After the authentication process is successful, the QuantumNexusLink instance, which more specifically is a link between the QuantumNexus and a QuantumEntity, is saved in the map of connected clients with the ID of the node as key, otherwise if the authentication process is failing, the NetGuard component is notified. The authentication system will be more detailed in the next chapter, called "Implementation", at the corresponding section for this system.

When a connection request passes the authentication and is added in the connected clients map, the server can start to send requests to the entity that just connected. The types of requests will be explained in the QuantumExchange section.

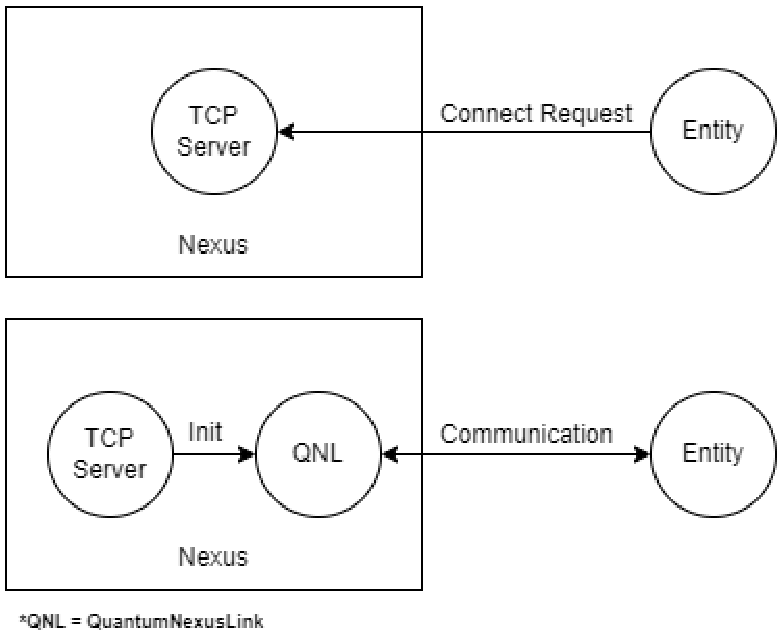


Figure 4: QuantumNexusLink Component

4.1.2 QuantumExchange Sub-Component

The Nexus Component's role is to handle the Application Programming Interface (API), which is made accessible over a secure websocket server. There are several predefined packets that can be received through this websocket. These packets are processed within this component and then forwarded to the associated QuantumNexusLink based on the received nodeID. There are two categories for this packets, the one initiated by this component and the one which are coming as a request for this component.

The predefined requests which can be processed over this websocket are:

- Task Execution Request
- Task Execution Stop Request
- Metrics Request
- Metrics Stop Request

The second category of packets contain a single type of message, which one is the authentication packet, which validate if a node exist in the database, this is why this packet is different, because is sent, not received, from QuantumExchange over the API.

The defined contents of these packets can be seen in the following diagrams represented in Figure 5.

Each request received by the server will generate a response packet that includes the TaskID and a boolean value indicating the success or failure of the request processing.

The single request made by the server, the one with the authentication packet, will receive as response the NodeID from the database.

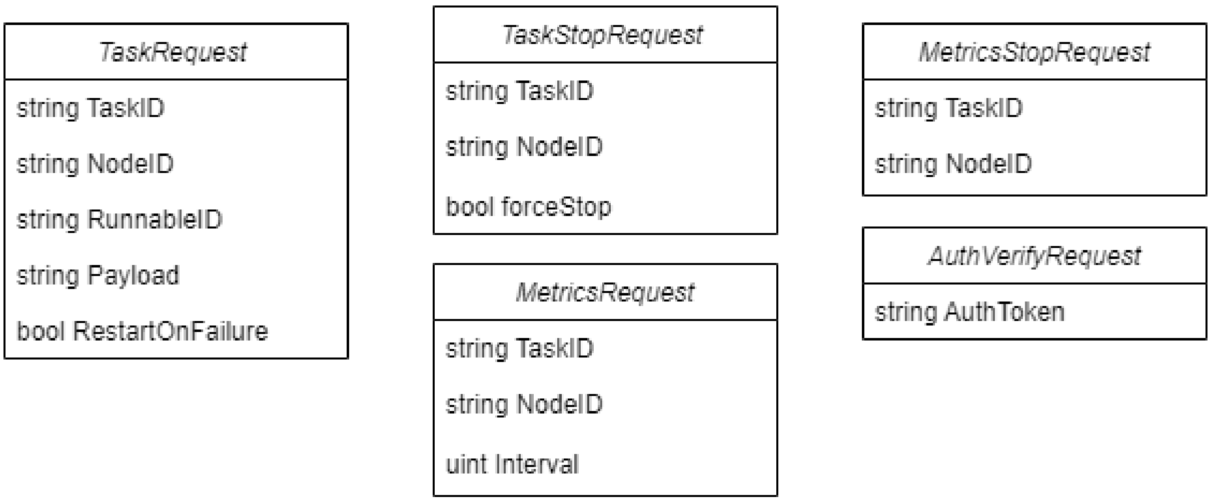


Figure 5: API Packets

4.1.3 SynapticCache Sub-Component

The purpose of this component is to reduce the failure rate of the process execution on compute nodes. To achieve this, the component have an internal map, where stores for every NodeID a list with the active jobs.

At every task request received through the API, if the boolean flag "RestartOnFailure" is true, then the task will be added to the map with the designated NodeID key. The task will be removed from the list of active tasks, when QuantumNexusLink will receive a notification packet from the Entity. To prevent the race condition, since every QuantumNexusLink runs on it's own thread, the component implements the add and remove functions which are thread-safe using synchronisation primitives.

In the event of node failure, when the node is back online, all the unfinished jobs will be re-send to execution.

In the case when the server is closed, first it will serialize and save the map into an encrypted file. At startup, if the file is available, the map will be loaded from it.

4.1.4 NetGuard Sub-Component

This component's function is to avoid both Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks. It implements internally a wrapper over the Linux firewall utility, iptables[16].

In addition, it receives notifications from the QuantumNexusLink component when an IP address fails or succeeds. If a request is failing the authentication process more than 3 times, it will get banned using the iptables[16] wrapper for 1 hour. After the ban expires if the requests from the same IP address are still failing, at every 3 failed attempts the last time will be multiplied with 3. At a successful authentication the banned IP is deleted from the failed authentication list.

To unban an IP address after the penalty expires, it is implemented a "daemon" in a standalone thread, which checks at established period of time if any rule of drop packets should be removed.

4.2 Entity Component

The second essential component is represented by the Entity component, which should run on a separate machine, and is the endpoint for receiving and executing jobs, being managed remote by the first essential component, the Nexus server.

To maintain a secure environment for the designated machine to run the Entity component, it doesn't need to use network address translation (NAT) method to expose the service to the wide area network (WAN). The paper [9] written by Haoyuan Wang, Yue Xue, Xuan Feng, Chao Zhou, and Xianghang Mi highlights that port forwarding can lead to substantial security vulnerabilities across the entire system and network. More precisely, the authors highlight that making ports accessible to the public network increases the chance of cyberattacks, including unauthorized entry, Denial of Service (DoS) attacks, and the exploitation of software vulnerabilities. So we choose to use port forwarding only one the machine that runs the main server (Figure 6), and in this case the security should be maintained only on this one, nodes machines being safe from this point of view.

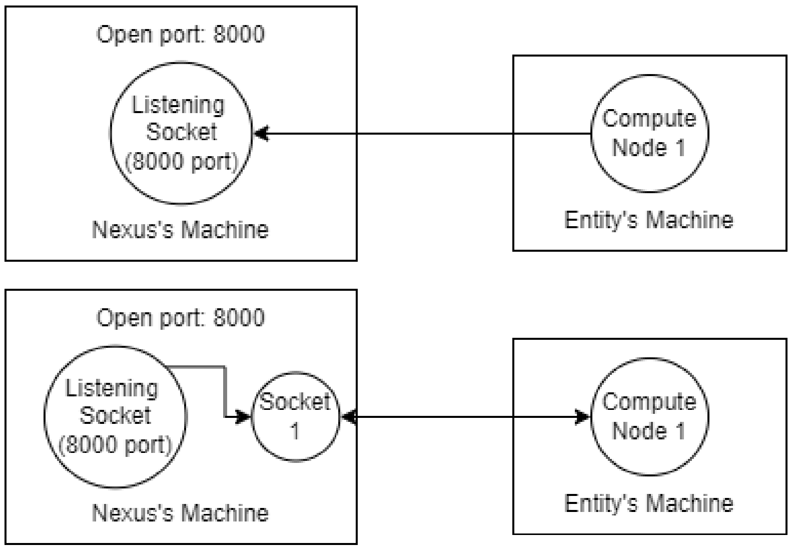


Figure 6: TCP Connection

On the entity component startup, it needs to load a config file (Listing 4.1), that should contain the authentication token generated in the web control panel, at node creation. This token it's the one used in the authentication process, which is covered in more detail in the next chapter.

```
[CONFIG]
authentication_token = XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX
```

Listing 4.1: Example of config.ini file

4.2.1 QuantumCalculia Sub-Component

This component is called from the main component, Entity, every time it receives a request of following types from the server: TaskRequest or TaskStopRequest.

When a TaskRequest packet is received, the QuantumCalculia component creates a new thread for this specific job, followed by the creation of a new process using the system calls fork() and execve() for the received executable path.

To get live data streams from stdin and stdout, 2 pipes are created, both doing the same job, but for a different data stream. The read ends of the pipes are on the QuantumCalculia thread and the write ends are on the new process side. At every data received on any of these pipes, Calculia thread will invoke the KafkaWrapper, which stores the data in the Kafka Broker[19] until it will be read in the web control panel, the KafkaWrapper will be explained in the following sections.

When a TaskStopRequest is received, the running thread of QuantumCalculia it's notified, and it sends a kill signal(SIGKILL) to the corresponding process, identified by process identifier (PID). If the received packet had the forceStop flag on true, the signal sent to the child process, will be SIGKILL.

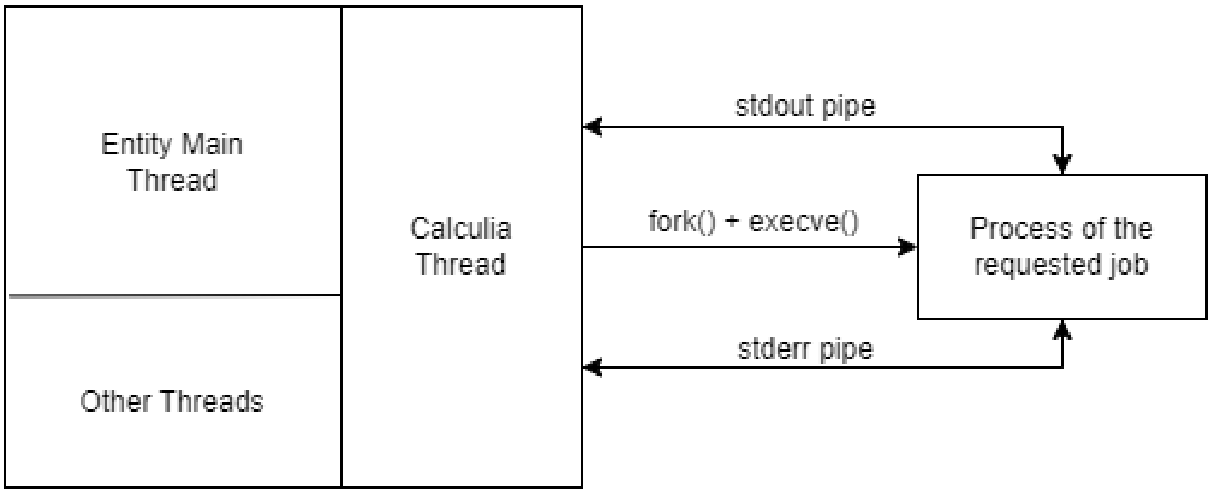


Figure 7: Process Execution

4.2.2 InsightPulse Sub-Component

This component implements a system for monitoring system metrics, such as CPU usage and memory usage. These data are read from the Linux interface for real-time system and process information, the virtual directory /proc.

For the CPU load the information is read from /proc/stat, then it's calculated the CPU total time and the CPU idle time and it's computed the CPU load. For the memory the information is read from /proc/meminfo the TotalMemory and the FreeMemory data.

When a RequestMetrics packet is received, the InsightPulse starts a thread which read these metrics and send them through the KafkaWrapper at a fixed interval defined by the field Interval from the received packet.

When a RequestStopMetrics is received, the thread corresponding to the request is stopped.

4.2.3 KafkaWrapper Sub-Component

This component is essential for the Entity component because the other two components depend on it. Without this component the live data from stdin and stdout streams and the live metrics can't be delivered to the web control panel. The three specified streams each have their own topic in the Kafka Broker[19], as shown in Table 1.

KafkaWrapper requires a Kafka Broker[19] for it to work, which can be on the same system as the main server or on a separate one. Kafka[19] is a fault-tolerant solution that ensures the reliable handling and delivery of essential streams required for monitoring distributed systems. Using Kafka's high throughput and low latency features, we can log real-time data across our infrastructure.

Stream	**Path**
STDOUT	{KAFKA_BROKER_URL}/{TASK_ID}/stdout
STDERR	{KAFKA_BROKER_URL}/{TASK_ID}/stderr
METRICS	{KAFKA_BROKER_URL}/{TASK_ID}/metrics

Table 1: Kafka Stream Paths

4.3 Common Components

The following components are used by both the Nexus and the Entity, each playing an essential role in this project. These components are fundamental to the system's architecture and functionality, ensuring efficient operation and integration across both main components. They provide critical services such as data serialization, security, and system monitoring, which are vital for the the project. Detailed descriptions of their functions are provided in the following sections, highlighting how each component contributes to the overall system.

4.3.1 ParagonEnigma Sub-Component

This component plays a crucial role in the communication system of the infrastructure. The secret behind it, it's the serialization wrapper implemented on top of the MessagePack library, which is one of the fastest (in terms of execution) and smallest (in terms of serialized data) library.

All the packets are serialised before being sent over the network and deserialized after are received over the network, ensuring consistency, data reliability, and faster transfers.

The structure of a serialized message is a MessagePack array of 2 serialized elements, as represented in Figure 8. The first array element, is the MessageType which is an uint8_t data type, which stores predefined values from 0 to 255, corresponding to the defined message structures. The second parameter is exactly the message object associated with the MessageType field.

This type of structure was chosen for easy deserialization in C++[4], using template functions. The first step at deserialization is to get the message type, then a generic function with the corresponding type of parameters can be called.

Furthermore, this approach enhances the scalability and maintainability of the communication system, by utilizing a standardized message format, it becomes simpler to introduce new message types. Additionally, the efficiency of msgpack[8] ensures minimal overhead, allowing the system to handle a high volume of messages with low latency, more details about this library in the next chapter.

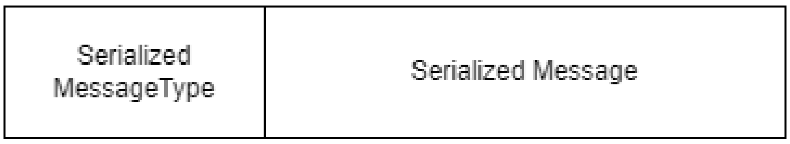


Figure 8: Packet Structure

4.3.2 QuantumInspector Sub-Component

This component is very important for logging and debugging, implementing multiple levels, depending on the current stage of the software.

QuantumInspector log data into an output file and must be instantiated first in the main function to ensure all data is logged through it. It implements a singleton pattern, to get the same instance from main in all components of the project. Due to, the multi-threading implemented for various components, it implements also a thread-safe manner to log data into the output file.

The defined types of logging available in project are:

- Notify Message - used for notifications, such as node successfully authenticated
- Error Message - used for every error in any component
- Debug Message - used only in software development stage
- Sent Message - used only in software development stage
- Received Message - used only in software development stage

4.3.3 QuantumSentinel Sub-Component

The QuantumSentinel it's the most important component of the entire project, being responsible with the entire cryptographic part of the communication (encryption and authentication).

This component is implemented of top of cryptographic library, MbedTLS[24], being the optimal one for this project requirements, which need to run on any device from embedded devices to super-computers. It integrates from this library the symmetric key exchange, message encryption, message decryption, digital certificate signing, and digital certificate verification.

The functionalities of this component are crucial and are used the best available techniques from the library:

- The symmetric key exchange is implemented using the Elliptic-curve Diffie–Hellman[13] (ECDH) algorithm and SHA-256[14] algorithm for hashing. The usage of these two algorithms ensures the confidentiality, integrity, and authenticity of the symmetric key exchange process, protecting against eavesdropping, man-in-the-middle attacks, and unauthorized access to data.
- The symmetric key used for encryption and decryption is an AES-256[12], derived from the previous step using the ECDH[13] algorithm, which ensures secure and confidential communication through robust key exchange and protection against unauthorized decryption attempts.
- Digital certificate signing and verification use RSA-4096[17] public and private key cryptography, ensuring robust security for authenticating identities and maintaining data integrity in secure communications.

4.3.4 SynchronizedThread Sub-Component

The SynchronizedThread component was designed to safely manage a background thread that performs periodic tasks. This component is used over all the components of the project, because it implements a safe and synchronized stop of the thread, made by the owner of the object.

The tasks are run into a new thread, which have a built-in functionality to sleep for a certain defined period, but even if it's sleeping, it can be stopped using a combination of an atomic boolean and a condition variable. Moreover, if a task wasn't joined and stopped, when the destructor of this component is called, the thread will be stopped in safe way.

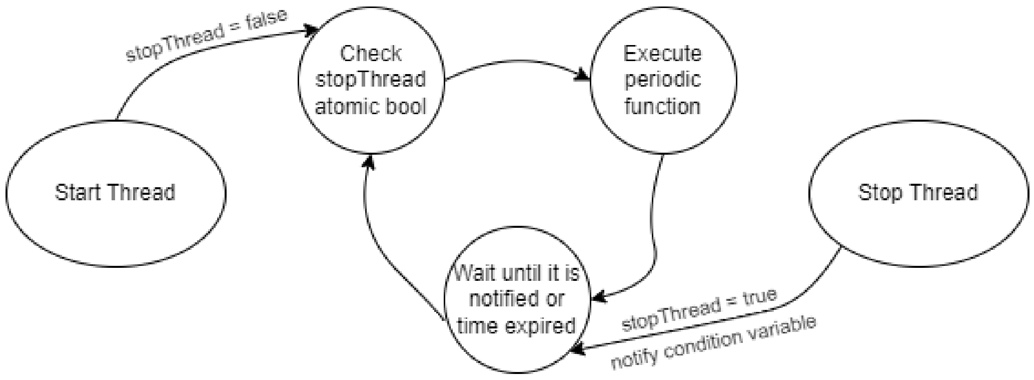

stopThread = false
Check stopThread atomic bool
Execute periodic function
Start Thread
Stop Thread
Wait until it is notified or time expired
stopThread = true
notify condition variable

Figure 9: Thread Execution Steps

4.3.5 Utilities Sub-Component

The utilities component includes various utility functions, that are used in multiple components and supports data processing and error handling, such as string manipulation, byte conversion, time management and network error handling.

These utility functions are designed to simplify basic programming tasks, as the ones presented previously. This component reduces the complexity of code and enhances its readability. Following the code modularity and reusability, we are ensuring that all the components use the same functionality for a certain task.

5 IMPLEMENTATION

In the previous chapter of this thesis, was presented an overview of all project's functionalities more or less in a non-technical way. This chapter will provide a more comprehensive analysis of the fundamental features and their detailed implementation. Through careful explanation of each component, the objective is to gain a better understanding of how these functionalities and how they contribute to the overall system architecture.

5.1 Data Serialization

Data serialization is a vital functionality for this project, ensuring data integrity and enabling faster communication between components over network, which is a major part of the project.

To achieve an efficient serialization, we choose MessagePack as our library. It acts like JSON[7], helping to exchange data across multiple languages and components over network, but it's smaller and faster. It is available in any programming language and it's used by a lot of big companies from tech industry, with all of these benefits we gave it a chance and it works as expected in our project.

Before choosing a library for serialization, I made some tests for both MessagePack and JSON[7], to choose the best one. From the tests I made, the size of the packets serialized with MessagePack in most of the cases were with more than 50% lighter than the one serialized with JSON[7], as it is presented in the Table 2.

Type	MessagePack	JSON
Null	c0 (1 byte)	null (4 bytes)
Integer	1B (1 byte)	27 (2 bytes)
Array (3 Integers)	93 0E 25 1D (4 bytes)	[14, 37, 29] (12 bytes)
String (5 Chars)	A5 48 65 6C 6C 6F (6 bytes)	"Hello" (7 bytes)
Map (1 Entry)	81 A4 54 79 70 65 C0 (7 bytes)	{"Type": null} (14 bytes)

Table 2: MessagePack vs JSON size comparison

5.2 Cryptography

As the previous functionality, the Cryptography is also a vital one, even a critical one for the entire system. The entire system of communication over network is based on the cryptography component, to ensure data integrity and security.

To comply with the presented requirements, we searched for the best library that we can use, which is also a lite one to run on embedded devices, but at the same time to offer the best security. For C++[4] exist a lot of libraries for for cryptographic operations and secure communication, such as mbedTLS, OpenSSL, Crypto++, libsodium and much more. For us the best options was mbedTLS, because it's a lightweight and efficient library and it's easy to integrate.

Mbed TLS[24] is an open-source library that offers a comprehensive implementation of cryptographic primitives, certificates management, and the SSL/TLS and DTLS protocols. The library supports the modern cryptographic standards through the PSA cryptographic APIs. This comes with a benefit for us, running on every devices, even on embedded ones, where resource efficiency is critical.

5.2.1 Secure Key Exchange

Using a symmetric key to encrypt communication, requires a safe key exchange algorithm, to really protect the key and then the entire communication system against different cybersecurity attacks.

The first implementation of the key exchange was through a encrypted connection using asymmetric encryption. The nodes had the server public key in their executable and the server get the node public key through the web control panel and the websocket communication system. After some detailed research, I found that it's not the best solution to exchange the symmetric key, because if some hacker or even a governmental agency run a eavesdropping attack over the communication and one day they get the private key of the server, they can decrypt all the packets where the symmetric key for a session was send and then with it they can decrypt the rest of the packets.

The second implementation, and the one is used right now, is using Elliptic-curve Diffie–Hellman[13] (ECDH) which functionality is only to generate symmetric keys in a secure way on two devices which communicate in a non-encrypted channel. This method ensures that the keys are never transmitted over the network and can't be found in any way using the current available computational power, thereby it significantly enhance the security of the key exchange process.

5.2.2 Message Encryption

The message encryption functionality is based on mbedTLS[24] symmetric encryption functions. The first step in the process of encrypt is the generating of the unique initialization vector (IV) based on the size of the plain text and a random generator from the library. After that the IV and the secret key are hashed multiple times, to generate a parameter used in context of AES[12] context initialization, that increase the security of the encryption.

In the next steps, the plain text is securely encrypted block by block. For each block of data, is computed a message authentication code (MAC) using Hash based Message Authentication Code (HMAC) to verify its authenticity and integrity at decryption.

After encrypting all the data blocks, it's appended the final message authentication code (MAC) to the encrypted data, which ensure that the encrypted data remains secure and unaltered during transmission.

5.2.3 Message Decryption

The decryption functionality is also based on mbedTLS[24] symmetric decryption capabilities. Initially, it verifies the integrity of the encrypted data by checking its length and ensuring the validity of the cipher block size. After this, the initialization vector (IV) and the secret key are hashed the same number of iterations like in the encryption process.

After that, the decryption proceeds is made in the same way as the encryption in blocks, each block having a Hash-based Message Authentication Code (HMAC), which validates the authenticity and integrity of each block during the decryption process.

The last step, is to verify the final message authentication code (MAC) appended to the encrypted data. This verification ensures that the decrypted data is unaltered throughout the transmission over network.

5.2.4 Digital Certificates

Digital certificates are a vital part of modern cryptographic systems and are used to ensure data integrity and authenticity. The implementation of the sign and verify of these certificates is made using mbedTLS RSA[17] certification functionality. The padding scheme used is PKCS #1 v2.1[17] (also known as RSA-PSS), which introduces the probabilistic signature scheme, which adds randomness to the padding process, enhancing security against certain cryptographic attacks.

5.3 Entity Authentication

At initialization of the authentication process, both Nexus and Entity generate their own public and private keys, which are represented by some large numbers generated using mbedTLS ECDH[13] functionality, then the Entity starts the communication sending own public key to the Nexus, followed by a order reverse, where the Nexus is sending the public key with a digital signed certificate to the Entity. After this step, both parts can generate the symmetric key, which can be used for the encrypted connection using AES-256[12].

Following the symmetric key exchange, the Entity sends the authToken, read from the config file (this functionality was described in the previous chapter). After Nexus receives the auth-Token over the encrypted channel with the negotiated symmetric key, it sends the token to the web control panel, to request a validation for it. If the token is generated by the panel and it's valid, the Nexus will receive on the websocket the node ID, otherwise it will receive the ID 0, which doesn't belongs to any node, and it's defined as invalid token. The entity then verifies the digital signature of the server, and if it is compromised, then will close the connection to the server.

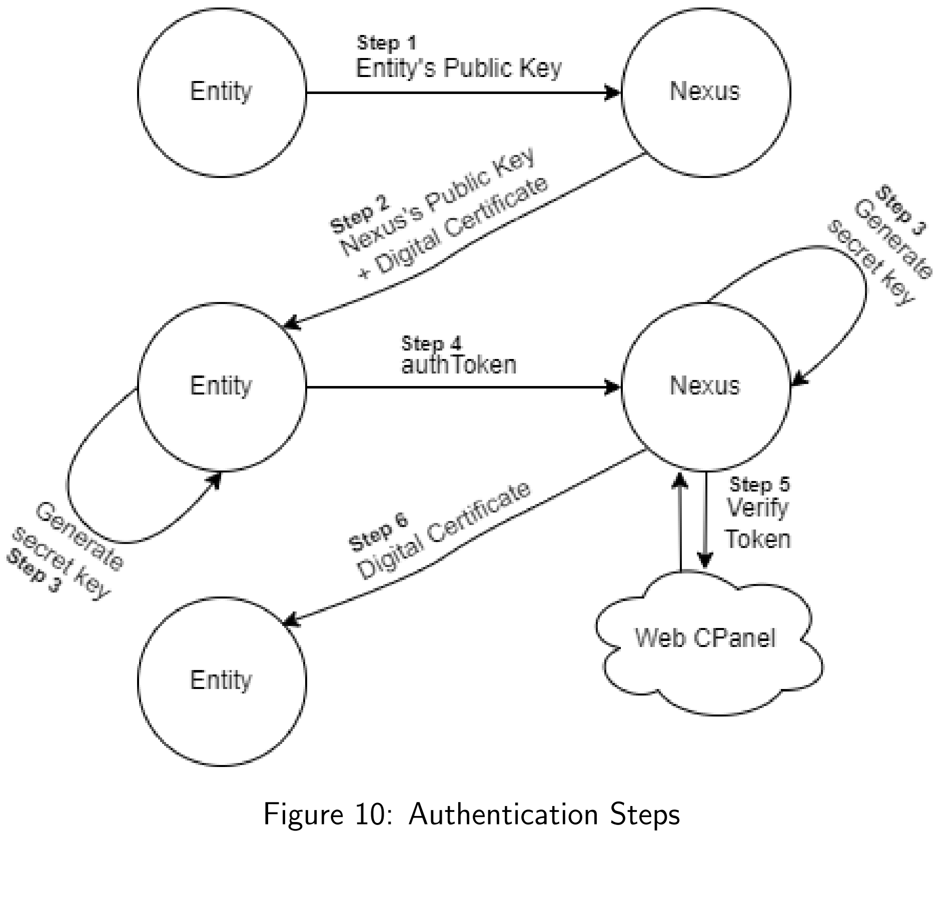


Figure 10: Authentication Steps

5.4 WebSocket API

The application programming interface (API) is exposed over a websocket server to facilitate real-time communication between the server and the web application. This API supports different types of requests and responses, as explained in QuantumExchange Component section. To offer an execution from the main component of the server, the QuantumExchange integrates observable queues for any response type to execute custom handling functions, providing a robust solution for asynchronous message processing.

The case when a request was received was solved with observable queues, that execute a custom function which will return a response over the websocket, but the case when the server make a request and is waiting for a response is more complicated, because the execution of the current thread should stop until the response is received. The use of std::future[4] and std::promise[4] compound with the observable queue facilitates the handling of asynchronous tasks and responses between the client and web application. Promise is used on the producer side, in the handle function from the observable queue, where the response will be set and the future is used on the consumer side, where the request was send and is used to wait until a response is received.

The integration of observable queue, std::future[4] and std::promise[4] create a robust mechanism for handling real-time asynchronous communication, by enabling non-blocking operations and ensuring thread safety, which will result in a responsive system.

5.5 Memory Handling

Efficient memory management is a cornerstone of high-performance systems, especially in real-time environments. In C++[4] the memory can be managed using advanced techniques, who enhance the reliability and efficiency of any software. Instead of using raw pointers, this project is using smart pointers[4], which automate the process of memory allocation and deallocation, minimizing the risk of memory leaks and dangling pointers. Using these smart pointers[4], our application achieves better resource utilization and also ensures that memory is correctly and safely managed throughout its lifecycle.

5.6 Apache Kafka

Apache Kafka[19] is the solution used by thousands of companies, which needs high-throughput and low latency, making it ideal for our project, which require faster and reliable real-time communication for metrics and live output from process execution. This product is a distributed streaming platform, very high scalable and fault-tolerant, which made it the perfect match for our requirements:

- Scalability - architecture allow for horizontal scaling
- Durability and Fault Tolerance - replicate messages across multiple brokers
- High Throughput and Low Latency - can handle high data volumes in real-time
- Flexible Message Processing - supports both real-time and batch processing

5.7 Compile and Deployment

For any modern software, containerization and continuous integration/continuous deployment (CI/CD) are essential. To achieve this, we are using a set of 3 important tools: cmake, docker and GitHub actions.

Both main components, Nexus and Entity, had their own Dockerfile, where multiple rules are set from files copy, to executable compiling using cmake and then copy the executable and reinstall the image of Linux to overwrite the source code.

To automate this docker image creation, we are using GitHub Actions, which detect every pull request that is approved to be merged in main branch, create the docker image and then upload it to the docker repository.

6 EVALUATION

The implementation of the proposed solution met all the requirements discussed with our possible investors and clients for the minimum viable product. Our next steps are include optimizing and refining all the components to ensure maximum performance and reliability. After that stage, we will continue to develop and integrate new features based on feedback and market demand.

To evaluate our current stage of development, we select a series of tests that will evaluate our crucial components, including the encryption system, anti-flooding system, and communication system. In order to accomplish this, we use a variety of tools, including: Tcpdump, Wireshark, Nmap, Iperf, Postman and multiple devices, both physical and cloud virtual machines.

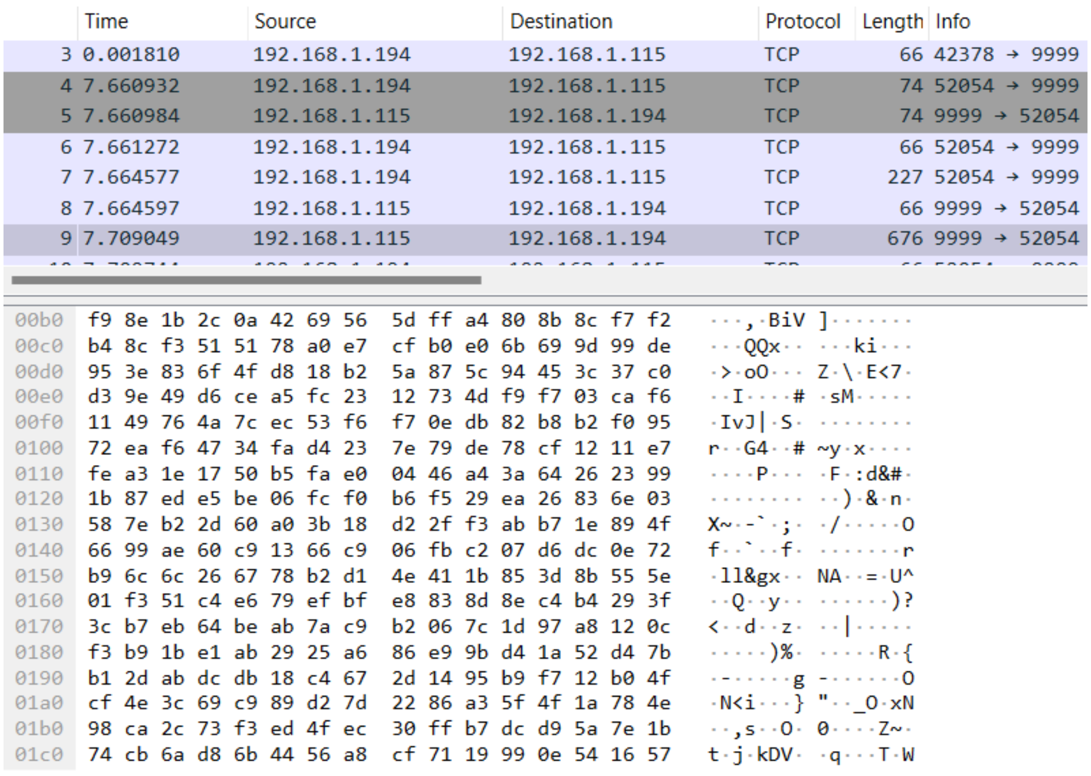
For the encryption system, the tests were performed to simulate interception attacks, such Man-in-the-Middle (MitM), packet sniffing, and replay attacks. The objective was to analyze the raw data received from TCP[10] sockets in order determine if the data could be interpreted as plain text or decoded using our serialization library, MessagePack.

For the purpose to evaluate the anti-flooding system, tests were performed by sending random messages containing random data packets using the Nping tool. The goal was to evaluate the system's capacity to prevent Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. This was accomplished by creating and implementing automatic iptables[16] filters to efficiently identify and prevent malicious network traffic.

For the communication system, tests were conducted to analyze the response time taken by our infrastructure for task execution requests across multiple environments. The objective was to measure and compare the efficiency and performance of our system in different scenarios to ensure optimal functionality.

6.1 Encryption System

As previously described, to test this component, I simulated packet sniffing attacks, using Tcpdump to capture packets on the server machine and Wireshark to analyze their content. The analysis, as shown in Figure 12, revealed that the data was unreadable, indicating successful encryption using the symmetric AES[12] algorithm. Attempts to interpret the data with our serialization library, MessagePack, were unsuccessful and the data remained unreadable without the encryption key, which is securely exchanged using our authentication system.



```
     Time         Source           Destination      Protocol  Length Info
   3 0.001810     192.168.1.194    192.168.1.115    TCP           66 42378 → 9999
   4 7.660932     192.168.1.194    192.168.1.115    TCP           74 52054 → 9999
   5 7.660984     192.168.1.115    192.168.1.194    TCP           74 9999 → 52054
   6 7.661272     192.168.1.194    192.168.1.115    TCP           66 52054 → 9999
   7 7.664577     192.168.1.194    192.168.1.115    TCP          227 52054 → 9999
   8 7.664597     192.168.1.115    192.168.1.194    TCP           66 9999 → 52054
   9 7.709049     192.168.1.115    192.168.1.194    TCP          676 9999 → 52054
```

```
00b0  f9 8e 1b 2c 0a 42 69 56  5d ff a4 80 8b 8c f7 f2   ···,·BiV ]·······
00c0  b4 8c f3 51 51 78 a0 e7  cf b0 e0 6b 69 9d 99 de   ···QQx·· ···ki···
00d0  95 3e 83 6f 4f d8 18 b2  5a 87 5c 94 45 3c 37 c0   ·>·oO··· Z·\·E<7·
00e0  d3 9e 49 d6 ce a5 fc 23  12 73 4d f9 f7 03 ca f6   ··I····# ·sM·····
00f0  11 49 76 4a 7c ec 53 f6  f7 0e db 82 b8 b2 f0 95   ·IvJ|·S· ········
0100  72 ea f6 47 34 fa d4 23  7e 79 de 78 cf 12 11 e7   r··G4··# ~y·x····
0110  fe a3 1e 17 50 b5 fa e0  04 46 a4 3a 64 26 23 99   ····P··· ·F·:d&#·
0120  1b 87 ed e5 be 06 fc f0  b6 f5 29 ea 26 83 6e 03   ········ ··)·&·n·
0130  58 7e b2 2d 60 a0 3b 18  d2 2f f3 ab b7 1e 89 4f   X~·-`·;· ·/·····O
0140  66 99 ae 60 c9 13 66 c9  06 fb c2 07 d6 dc 0e 72   f··`··f· ·······r
0150  b9 6c 6c 26 67 78 b2 d1  4e 41 1b 85 3d 8b 55 5e   ·ll&gx·· NA··=·U^
0160  01 f3 51 c4 e6 79 ef bf  e8 83 8d 8e c4 b4 29 3f   ··Q··y·· ······)?
0170  3c b7 eb 64 be ab 7a c9  b2 06 7c 1d 97 a8 12 0c   <··d··z· ··|·····
0180  f3 b9 1b e1 ab 29 25 a6  86 e9 9b d4 1a 52 d4 7b   ·····)%· ·····R·{
0190  b1 2d ab dc db 18 c4 67  2d 14 95 b9 f7 12 b0 4f   ·-·····g -······O
01a0  cf 4e 3c 69 c9 89 d2 7d  22 86 a3 5f 4f 1a 78 4e   ·N<i···} "··_O·xN
01b0  98 ca 2c 73 f3 ed 4f ec  30 ff b7 dc d9 5a 7e 1b   ··,s··O· 0····Z~·
01c0  74 cb 6a d8 6b 44 56 a8  cf 71 19 99 0e 54 16 57   t·j·kDV· ·q···T·W
```

Figure 11: Intercepted packets

The tests were successfully passed, and the component showed a strong defense against packet sniffing attacks. The encrypted data remained secure and unreadable without the ephemeral encryption key.

6.2 Anti-flooding System

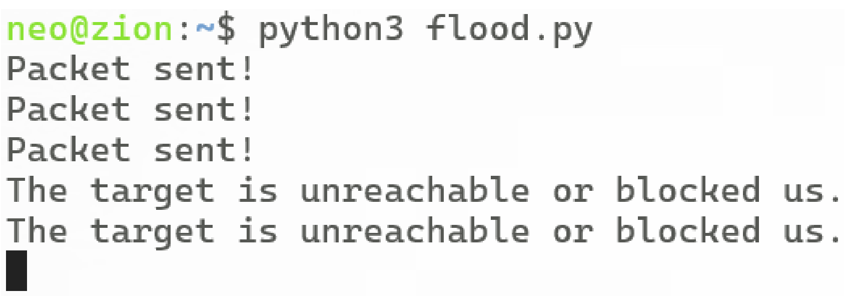
To test the NetGuard component described in Section 4.1.4, I generate a high volume of TCP[10] traffic, replicating the behaviour of a Denial-of-Service (DoS) attack, using the Python script presented in Listing 6.1.

```python
def send_packet(target_ip, target_port, message):
    while True:
        try:
            sock = socket.socket(socket.AF_INET, socket.
SOCK_STREAM)
            sock.settimeout(5)
            sock.connect((target_ip, target_port))
            sock.sendall(message.encode('utf-8'))
            sock.close()
            time.sleep(1)
            print("Packet sent!")

        except socket.timeout:
            print("The target is unreachable or blocked us.")
        except socket.error as e:
            print(f"Socket error: {e}")
```

Listing 6.1: Denial-of-Service script

By sending a continuous stream of packets to the server machine, it was possible to observe how the NetGuard Component handled the incoming flood of data, dropping all the connections from source IP of the DoS attack after 3 unsuccessful authentication, as shown in the Figure 12.



```
neo@zion:~$ python3 flood.py
Packet sent!
Packet sent!
Packet sent!
The target is unreachable or blocked us.
The target is unreachable or blocked us.
```

Figure 12: Anti-flooding system results

This test provided a detailed analysis of the system response in case of a DoS attack and how effective is it in filtering out malicious traffic while allowing legitimate data to pass through.

6.3 Communication System

For this stage of testing, we used multiple machines to evaluate our components in various environments. Each machine was configured differently to simulate a range of operating conditions, including variations in hardware specifications, network speeds, and software configurations. In doing this, we were able to gather comprehensive data on how our communication system performs under different circumstances. The server is running on Machine 1 and on every machine presented in the Table 3 is running a node. The results can be seen in Table 4.

Name	Specs	OS
Machine 1	i7-9750h, 24GB RAM	Ubuntu 23.10
Machine 2	Celeron N5095, 5GB RAM	Ubuntu 22.04
Machine 3	i7-1370P, 32GB RAM	Windows 11
Oracle Cloud	Emerald 5th, 1GB RAM	Ubuntu 22.04
Oracle Cloud ARM	Ampere A1, 24GB RAM	Ubuntu 22.04

Table 3: Machines specifications

*Machine2 is running with Proxmox as Hypervisor with 2/4 cores

*Oracle Cloud has 1 core

*Oracle Cloud ARM has 4 ARM cores

** Machine1, Machine2, and Machine3 are running on the same LAN

To make this test possible over the communication system, a task request using sleep utility tool for 30 seconds was send to every node, and then from the total time of the response is reduced with 30 seconds, taken by task execution. The request is send through the web control panel backend over the websocket and the server process the received message and re-send it to the requested node.

Name	Speed	Real Time	Total Time
Machine 1	36.2 Gbps	10ms	60ms
Machine 2	815Mbps	20ms	70ms
Machine 3	897Mbps	40ms	90ms
Oracle Cloud	51Mbps	50ms	100ms
Oracle Cloud ARM	683Mbps	40ms	90ms

Table 4: Results

*Total time includes the connection from web control panel backend to the server, which takes around 50ms.

6.4 Comparison

Our critical components were supposed to a series of tests to determine if they function as expected and to evaluate their efficiency. The purpose of these tests is to guarantee that every component meets our performance requirements and operates reliably under various conditions. Through this examination of these components, we proposed we verified that doesn't exist any problems and they work at their highest efficiency.

For the encryption system, the tests involved subjecting the system to a packet sniffing attack. During these tests, the analyzed packets were expected to remain unintelligible to both human observers and any analytical tools, demonstrating the effectiveness of the encryption.

The test for the anti-flooding system involved subjecting it to high traffic over the TCP[10] socket with garbage packets, simulating a denial-of-service attack intended to disrupt legitimate traffic through the infrastructure. During the testing, the component successfully blocked all the malicious traffic, and the infrastructure functioned as expected.

For the communication system, we conducted a series of task requests and carefully recorded the response times. This method helped us to identify potential bottlenecks and performance issues that could occur in real-world scenarios. The insights from these tests were crucial for fine-tuning our infrastructure, ensuring it can manage various deployment scenarious efficiently while maintaining high performance standards across all tested environments.

Finally, it passed all the tests and met all the requirements established with our potential clients and investors at the project ideas hackathon. It is now prepared for the further stages of development to become fully-fledged software.

7 CONCLUSIONS

The primary objective of this thesis was to perform a requirements analysis and subsequently to present the design and implementation of a modern software tool based on three core principles of software engineering: security, efficiency, and scalability. The tool is intended for both companies and individual users who need to automate the runtime of their applications in a secure remote environment with full control over data storage.

To achieve this, the tool creates a hosted on-premises environment acting as a private cloud, fully managed by the client without third-party intervention. The product targets two primary customer categories: companies requiring rigorous security compliance, such as those in banking, government, or healthcare, and IT enthusiasts.

The project targets the increasing demand for online services and the requirement for reliable infrastructure. It provides a self-hosted environment that combines the benefits of cloud computing with advanced control and security, providing an alternative to public cloud services.

The implementation involved developing two main components: the server (Nexus) and the compute nodes (Entities). The server acts as the orchestrator, managing the nodes and facilitating communication between them and the web control panel. The compute nodes execute the tasks assigned by the server.

Some of the key features include secure end-to-end encrypted communication using the mbedTLS[24] library, a strong anti-flooding system to prevent DoS attacks, and a real-time monitoring and statistics system. The tool is designed to run cross-platform, ensuring wide compatibility from embedded devices to supercomputers, regardless the operating system.

The results of the project were evaluated through a series of tests focusing on performance, security, and reliability. The encryption system successfully protected data from packet sniffing attacks, the anti-flooding system effectively blocked malicious traffic, and the communication system demonstrated high efficiency and low latency across different environments.

Overall, the project met all the requirements established at beginning with our potential clients and investors during the project ideas hackathon and is now ready for the next steps in development to became a fully-fledged software solution.

7.1 Future Developments

Potential future developments include:

- **Advanced Analysis**: Implement machine learning algorithms to analyze permanent the nodes and to choose the best one for every task, depending on multiple criteria, such as: node performance during similar tasks or node workload.
- **Full Compatibility**: Ensure full compatibility to run native the Entities on machines with Windows, without a docker image, adding the possibility to manage the operating system.
- **Scalability Enhancements**: Develop features that allow greater scalability, enabling support for larger and more complex applications.
- **Interoperability**: Ensure seamless integration with various third-party services and platforms to enhance functionality and user experience.
- **Recovery System**: Develop a robust disaster recovery system and backup solutions to protect user data against loss.
- **Certifications**: Obtain relevant certifications (e.g., ISO, HIPAA) to assure users of the platform's compliance with industry standards.
- **Community and Support**: Build a community forum, where clients can discuss and we can offer comprehensive support to help users troubleshoot their issues.

BIBLIOGRAPHY

[1] Inc. Amazon Web Services. Amazon web services (aws). https://aws.amazon.com/documentation/. Last accessed: 16 June 2024.

[2] Luca Cerniglia, Silvia Cimino, Eleonora Marzilli, and Giulia Ballarotto. How does psychological distress due to the covid-19 pandemic impact on internet addiction and instagram addiction in emerging adults? https://www.researchgate.net/publication/355738560_How_Does_Psychological_Distress_Due_to_the_COVID-19_Pandemic_Impact_on_Internet_Addiction_and_Instagram_Addiction_in_Emerging_Adults. Last accessed: 7 June 2024.

[3] Google Cloud. Google cloud platform (gcp). https://cloud.google.com/docs/. Last accessed: 16 June 2024.

[4] ISO/IEC JTC 1/SC 22/WG 21 The C++ Standards Committee. C++ standard library documentation. https://en.cppreference.com/. Last accessed: 22 June 2024.

[5] Microsoft Corporation. Microsoft azure. https://docs.microsoft.com/en-us/azure/. Last accessed: 16 June 2024.

[6] Oracle Corporation. Oracle cloud. https://docs.oracle.com/en/cloud/. Last accessed: 16 June 2024.

[7] Douglas Crockford. Javascript object notation (json). https://www.rfc-editor.org/rfc/rfc8259. Last accessed: 21 June 2024.

[8] Sadayuki Furuhashi. Messagepack. https://msgpack.org/index.html/. Last accessed: 18 June 2024.

[9] Xuan Feng Chao Zhou Haoyuan Wang, Yue Xue and Xianghang Mi. Port forwarding services are forwarding security risks. https://arxiv.org/abs/2403.16060. Last accessed: 10 June 2024.

[10] Internet Engineering Task Force (IETF). Transmission control protocol (tcp). https://datatracker.ietf.org/doc/html/rfc793/. Last accessed: 22 June 2024.

[11] Internet Engineering Task Force (IETF). The transport layer security (tls) protocol version 1.3. https://tools.ietf.org/html/rfc8446. Last accessed: 22 June 2024.

[12] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. Last accessed: 19 June 2024.

[13] National Institute of Standards and Technology (NIST). Elliptic-curve diffie–hellman (ecdh). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Last accessed: 17 June 2024.

[14] National Institute of Standards and Technology (NIST). Secure hash standard (shs). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. Last accessed: 20 June 2024.

[15] Ani Petrosyan. Global number of internet users 2005-2023. https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/. Last accessed: 7 June 2024.

[16] Netfilter Project. iptables. https://netfilter.org/projects/iptables/index.html/. Last accessed: 19 June 2024.

[17] RSA Security. Pkcs: Rsa cryptography specifications version 2.1. https://www.rfc-editor.org/rfc/rfc3447. Last accessed: 23 June 2024.

[18] Ansible Project Team. Ansible. https://docs.ansible.com/. Last accessed: 16 June 2024.

[19] Apache Kafka Project Team. Apache kafka. https://kafka.apache.org/documentation/. Last accessed: 16 June 2024.

[20] Chef Project Team. Chef. https://docs.chef.io/. Last accessed: 16 June 2024.

[21] Puppet Project Team. Puppet. https://www.puppet.com/docs/puppet/. Last accessed: 16 June 2024.

[22] SaltStack Project Team. Saltstack. https://docs.saltproject.io/. Last accessed: 16 June 2024.

[23] Bergur Thormundsson. Worldwide generative ai revenue 2020-2032. https://www.statista.com/statistics/1417151/generative-ai-revenue-worldwide/. Last accessed: 7 June 2024.

[24] TrustedFirmware. Mbed tls. https://mbed-tls.readthedocs.io/. Last accessed: 16 June 2024.

[25] Lionel Sujay Vailshery. Public cloud services end-user spending worldwide from 2017 to 2024. https://www.statista.com/statistics/273818/global-revenue-generated-with-cloud-computing-since-2009/. Last accessed: 7 June 2024.