Rapport de Stage M2

Développement de composants d'une plateforme d'orchestration de services composites dans le domaine maritime

Muhammad AL QABBANI

17 Juillet 2023 - 20 Décembre 2023

Tuteur de stage:
Mme Soraya AIT-CHELLOUCHE, M. Yassine HADJADJ-AOUL, M. Guillaume LE GALL

Superviseur Académique:
M. César VIHO

Établissement:
Université de Rennes - Master en Informatique Parcours Cloud et Réseaux

Établissement d'accueil:
IRISA, INRIA de l'Université de Rennes, France

Année universitaire: 2023/2024

La couverture de ce rapport a été créée en utilisant le créateur d'images AI de Microsoft Bing, mettant en vedette DALL-E 3

A Heartfelt Dedication

I dedicate this work to my cherished parents, beloved siblings, and my entire family, whose unwavering support and encouragement have been the cornerstone of my journey. Their steadfast belief in my capabilities has provided me with enduring strength. I am deeply thankful for their love, patience, and unwavering faith, as this achievement would not have been attainable without their presence in my life.

Acknowledgments

I am profoundly grateful to everyone who contributed to the successful completion of my internship. Firstly, I extend my heartfelt thanks to Mr. Cédric GUÉGUEN, the head of the M2 Informatique parcours CR (Cloud et Réseaux), for his exceptional commitment in overcoming administrative obstacles, ensuring a smooth continuation of my internship. Deep appreciation goes to Mr. Yassine Hadjadj-Aoul, my internship supervisor at IRISA/INRIA. His patience in providing me with the opportunity to undertake this internship has been truly remarkable. His unwavering support throughout my entire internship journey has been a constant source of motivation. Words seem inadequate to express my gratitude for his guidance and mentorship. My direct supervisor at IRISA/INRIA, Mme Soraya AIT-CHELLOUCHE, deserves special mention for her outstanding dedication and commitment in overcoming challenges, which were instrumental in facilitating my seamless integration into the internship. Her attention to detail, patience, and the diverse assistance she provided throughout my training period, such as valuable notes for presentations and for the internship report, were truly remarkable and deeply appreciated. I extend my deepest appreciation to the Ermine team at the IRISA lab for their warm reception and for facilitating the resources essential to my work. A special acknowledgment is reserved for Mr. Guillaume LE GALL, my mentor, whose unwavering support, collaboration, and instrumental guidance have proved invaluable. His extensive feedback on project-related inquiries and detailed notes significantly improved my defense presentation, a testament to his dedicated efforts across multiple sessions and hours dedicated to refining the content. I also acknowledge the guidance provided by my academic supervisor, Mr. César Viho. His constructive feedback during the NaviDEC project presentation significantly improved my presentation skills. Madame Gwenaelle Lannec, the diligent Team Assistant at IRISA, has been a pillar of support throughout my internship period. Her invaluable contributions have made a significant impact. I extend my sincere gratitude to the exceptional faculty at Rennes University, Master 2 cloud and network program, for their unwavering dedication and patience in imparting knowledge. Their commitment has guided us seamlessly through our academic journey. Lastly, I appreciate the administrative staff at the university for their tremendous efforts in resolving administrative challenges. Their dedication made our academic experience truly unforgettable. In conclusion, I extend my gratitude and heartfelt thanks to all for making my academic and internship journey at Rennes University, Master 2, a rich and rewarding experience.

For convenience, a PDF version of this report is available for download:
Rapport de Stage M2 CR Muhammad AL QABBANI IRISA Platform, NaviDEC.pdf
An HTML version can be accessed for browsing on this website:
Rapport de Stage M2 CR Muhammad AL QABBANI IRISA Platform, Website
A high-quality version of the diagram is also available for download:
IRISA Platform Diagram.svg

TABLE OF CONTENTS

CHAPTER

ONE

INTRODUCTION AND PROJECT OVERVIEW

1.1 Introduction

In this initial chapter, we will explore several key aspects that set the foundation for the rest of the report. The chapter is divided into three main sections. The first section provides an overview of the Work-Context, Accomplished Work, and Report Structure. This section will give you an insight into the internship experience at IRISA, the tasks accomplished during this period, and how this report is organized to present the information. Following this, we will introduce the NaviDEC Project, offering a brief introduction to the project. Finally, we will touch upon the Project Context and Use Cases, providing a snapshot of the project's operational context and its use cases.

1.1.1 Work-Context

This report presents the work carried out during my five-month internship at IRISA, a renowned research institute, as part of the Master2 program in Cloud and Network Infrastructure at the University of Rennes. The focus of my work was the development of components for a composite service orchestration platform in the maritime domain, a project of significant complexity and innovation. This period provided a unique opportunity to apply and expand my technical skills in a real-world setting.

1.1.2 Accomplished Work

During my internship, I accomplished approximately 12 tasks, each contributing to the development and enhancement of the IRISA platform. My journey began with understanding the existing codebase, developed a comprehensive logging system which laid the foundation for my subsequent work. The tasks ranged from infrastructure automation with Ansible Playbook, Docker container automation, to migrating the framework from Flask to FastAPI. Each task contributed to the project's progress and enriched my technical skills. Significant milestones of my internship included developing an interactive web-based terminal using xterm.js and creating the bandwidth visualization using Apache ECharts from scratch. I also used Sphinx to develop a user guide from scratch and published it on a website using GitLab Pages. The build and deployment process of the user guide was automated using GitLab CI/CD. The culmination of these tasks resulted in a robust platform capable of orchestrating composite services in the maritime domain. This platform is a testament to the hard work and dedication during my internship and is a significant contribution to the NaviDEC project.

1.1.3 Report Structure

This report is structured into three main chapters. The first chapter provides an overview of the project and the context in which it was carried out. The second chapter delves into the enhanced characteristics of the IRISA platform, detailing the various tasks executed and the challenges overcome. The final chapter explores the work organization methodology, providing a comprehensive account of the technical growth experienced during the internship. The report concludes with a reflection on the internship experience and the knowledge gained. Finally, this chapter includes a summary in English and a résumé in French of the work done.

1.2 Unveiling the NaviDEC Project

1.2.1 Introduction

This chapter provides an introduction to the NaviDEC Project, a Deep-Edge Computing Platform. We will discuss the project's objectives, partners, and the scope of work for the Proof of Concept at the IRISA Team. Additionally, we will delve into the internship working environment at IRISA and the ERMINE Team's structure and goals. The ultimate aim of the internship is to enhance the Testbed Platform's characteristics. Let's dive in!

Fig. 1: NaviDEC Project Logo.

1.2.2 About NaviDEC

The NaviDEC Project is a Deep-Edge Computing Platform financed by the region of Brittany. It operates as an "IoT and Image Processing" platform, extending Edge Computing to the deepest edge of the network. The primary objective is to enhance efficiency in processing and transmitting data in regions with intermittent connectivity. NaviDEC will undergo testing in maritime scenarios, deployed in both deep-edge (boat) and edge (land) gateways. The system is designed to meet the challenges of environments that require immediate data processing responses.

1.2.3 Project Timeline

The NaviDEC project commenced in November 2021 and is scheduled to continue until April 30, 2024. There is a proposal to extend the project until the end of 2024. This timeline allows for comprehensive research, development, and testing of the project's objectives and goals.

1.2.4 The Ultimate Project Goal

The ultimate goal of the project is to mitigate the risks of boat collisions by utilizing cameras for streaming and employing AI image processing, known as a classifier.

1.2.5 Broad Objectives of the Project

1. Define a "deep-edge" architecture and its interfaces with standard clouds across various use cases.
2. Study and develop distributed video processing algorithms spanning the deep edge and edge.
3. Investigate and develop service placement and orchestration strategies between the deep edge and the edge.
4. Research and develop a coordinated deep-edge security and anomaly detection system integrated with edge.
5. Deploy and test a prototype in real-world use cases that involve intensive data processing, particularly in the context of maritime scenarios.

1.2.6 Collaborating Entities

The NaviDEC project involves several partners, each contributing at various levels such as technical, financial, and managerial. These include Université de Rennes, Inria, IRISA, InPixal, Sodira, Céladon, Thalos, Region Bretagne, Rennes Metropole, bpifrance, Images & Réseaux, among others. In the following sections, we will highlight the most significant partners to provide a better understanding of the project context.

- **IRISA/INRIA:**

  Develop and integrate the platform on the boat (Deep-Edge) and land (Edge). During my entire internship period, I focused on enhancing this platform, which is the main area of my contribution.

- **Sodira:**

  Develops a multi-interface router for boat connectivity.

- **InPixal:**

  Handles video and image processing.

- **Celadon:**

  Provides the boat equipped with cameras for the final experience.

Fig. 2: NaviDEC Project Partners.

1.2.7 IRISA Team's Role in the Proof of Concept

The IRISA team's specific focus aligns with Objective 3 from the Broad Objectives of the NaviDEC Project. This objective is to investigate and develop service placement and orchestration strategies between the deep edge and the edge. Each partner has a stake in the project, and at IRISA, we have a defined scope of work. Our focus is on the Proof of Concept team.

The project at IRISA is progressing through three phases:

1. **Phase One: Proof of Concept:**

   This phase involved designing the IRISA platform in the lab. This phase has already been completed by a previous colleague and the responsibilities have since been handed over to me. I am now carrying forward the work that was initiated in this phase.

2. **Phase Two: Enhancing Platform Characteristics:**

   This is the current phase and the scope of my internship. In this phase, I am responsible for carrying out all the work, which includes refining and enhancing the features of the platform. This has recently been completed. The tasks accomplished during this phase will be detailed in the upcoming chapter.

3. **Phase Three: Integration:**

   This phase is dedicated to integrating the work accomplished in phases one and two with our related partners, applying it to real-world test fields. This phase will start following the completion of the internship period, and I will be responsible for executing this work. The aim is to ensure that the outcomes of the project are robust, reliable, and ready for deployment in real-world settings.

1.2.8 Concept of the Testbed Platform

The Testbed Platform is designed for the automated placement of service resources components, allowing them to move between different clusters. Specifically, in the context of NaviDEC project, the movement occurs between the Deep Edge (located on boat) and Edge (located on land) Computing Network, and it's based on the current bandwidth strength.

The idea revolves around two clusters. The first cluster hosts resource services like streaming and classifier deployment pods with its services and called the Deep Edge , which is located on the boat.

The placement of the classifier is bandwidth-dependent. When the bandwidth is strong, the classifier is automatically relocated from the first cluster on boat to the second cluster on land. Conversely, when the bandwidth is low, it is relocated back to the first cluster on boat. This dynamic relocation ensures optimal performance based on the current network conditions. Designing a Platform for Automated Service Execution Placement bridging the Deep Edge (boat) and Edge (land) Computing Network.

A classifier is used to process video streaming images and identify other boats using artificial intelligence. This AI is designed to identify other boats in the vicinity. The purpose of this system is to detect and prevent potential accidents. If the system identifies a boat that is too close to the ship, it triggers an alarm. This alarm serves as an early warning system, allowing the crew to take necessary actions to avoid any possible collisions at sea.

The computing power on the boat (Deep Edge) is limited because it uses an IoT platform. But on land (Edge), the CPU has more power for image processing. So, when the bandwidth is strong, we move the classifier to land to take advantage of this extra power. But when the bandwidth is weak, it moves it back to the boat. This way, we always get the best performance possible.

1.2.9 The Internship: Objectives and Expectations

The primary objective of the internship is to refine and enhance the features of the Testbed Platform. This improvement will pave the way for the next phase of the project, which involves integrating with related partners. This integration will be conducted in real-world scenarios using physical appliances, providing a practical and hands-on experience. This approach ensures that the project outcomes are robust, reliable, and ready for deployment in real-world settings.

1.2.10 Internship Working Environment at IRISA

Internship Organizational Structure Overview

My internship is being conducted at the IRISA laboratory at INRIA, University of Rennes, specifically within the ERMINE Team. Below are brief overviews of IRISA, the ERMINE Team, and the team structure.

IRISA: Advancing Computer Science and Innovation

IRISA, which stands for the Institute for Research in Computer Science and Random Systems, is a joint research unit. IRISA is today one of the largest French research laboratory (more than 916 people) in the field of computer science and information technologies. Structured into seven scientific departments, the laboratory is a research center of excellence with scientific priorities such as bioinformatics, systems security, new software architectures, virtual reality, big data analysis and artificial intelligence.

Located in Rennes, Lannion and Vannes, IRISA is at the heart of a rich regional ecosystem for research and innovation and is positioned as the reference in France with an internationally recognized expertise through numerous European contracts and international scientific collaborations.

Focused on the future of computer science and necessarily internationally oriented, IRISA is at the very heart of the digital transition of society and of innovation at the service of cybersecurity, health, environment and ecology, transport, robotics, energy, culture and artificial intelligence.

IRISA is a joint-venture resulting from the collaboration between nine institutions, in alphabetical order: CentraleSupélec, CNRS, ENS Rennes, IMT Atlantique, Inria, INSA Rennes, Inserm, Université Bretagne Sud, Université

de Rennes. From this collaboration is born a force that comes from women and men who give the best of themselves for fundamental and applied research, education, exchanges with other disciplines, transfer of know-how and technology, and scientific mediation.

ERMINE: Advancing Network Management and Economics

ERMINE Stands for mEasuRing and ManagIng Network operation and Economic. Networks are omnipresent and increasingly complex, and require an efficient management of their operations. The team designs and analyzes procedures and policies for efficiently managing network operations, but also works on the required measurement and monitoring of performance metrics. Our specific and original management activity focuses on network economics, regulation, and automated decision making. In terms of needed measurement, we make use of standard modeling and performance analysis techniques, but also carry out direct measurements to be analyzed statistically. A cross-cutting research axe for both management and measuring is artificial intelligence. Our activity is a trade-off between methodological/mathematical developments and practical implementations.

Industrial contacts

The team has historical relations with industrial partners on diverse areas in networking, including ALSTOM (on rare event simulation), Orange (on network economics, AI, wireless networks, network management), Nokia (on network economics, AI), Technicolor (on AI). It has also contacts with regulatory bodies on neutrality issues and the development of tools to monitor a satisfying behavior of actors of the Internet. We also have contacts with regulatory bodies.

As of 2022, the team is holding contracts with Nokia (Cifre and ADR; on several applications of Data Analysis and Machine Learning to networking), Exfo (Cifre; on reliable probes placement) and CISCO (Cifre; on localisation indoor, FTM5). The objective is to carry out common research on an integrated framework for 5G, programmable networks, IoT and clouds that aims at statically and dynamically managing and optimizing the 5G infrastructure using, in particular, Machine Learning techniques.

Team Members

Number of Persons:

At the ERMINE Team, there are currently approximately 20 individuals. This includes permanent researchers, emeritus members, research engineers, PhD students, and a team assistant.

Team leader:

- Mr. Bruno Tuffin

Internship supervisors:

- Mme Soraya Ait Chellouche
- M. Yassine Hadjadj-Aoul
- M. Guillaume LE GALL

Academic supervisor:

- M. César Viho

1.3 Project Context and Use Cases

1.3.1 NaviDEC Work Context

Consider a simple scenario where a boat is equipped with multiple cameras from Céladon for streaming and an onboard AI image processor, known as a classifier. The cameras capture photos, which are then analyzed by the AI image processor. If another boat or obstacle is detected in close proximity, it is highlighted on the output screen. Furthermore, when the distance between boats reaches a certain threshold, an alarm is triggered, serving as a signal to take immediate precautions to prevent potential collisions. This succinctly encapsulates the essence of the NaviDEC work context.

1.3.2 Illustration: NaviDEC Work Context

Below, an illustrative figure demonstrate the scenarios described above:

An illustrative representation of the NaviDEC work context, showcasing a boat equipped with cameras for streaming and an onboard AI image processor (classifier). The system captures and processes images, highlighting nearby boats on the output screen and triggering alarms at a specified distance to prevent potential collisions.

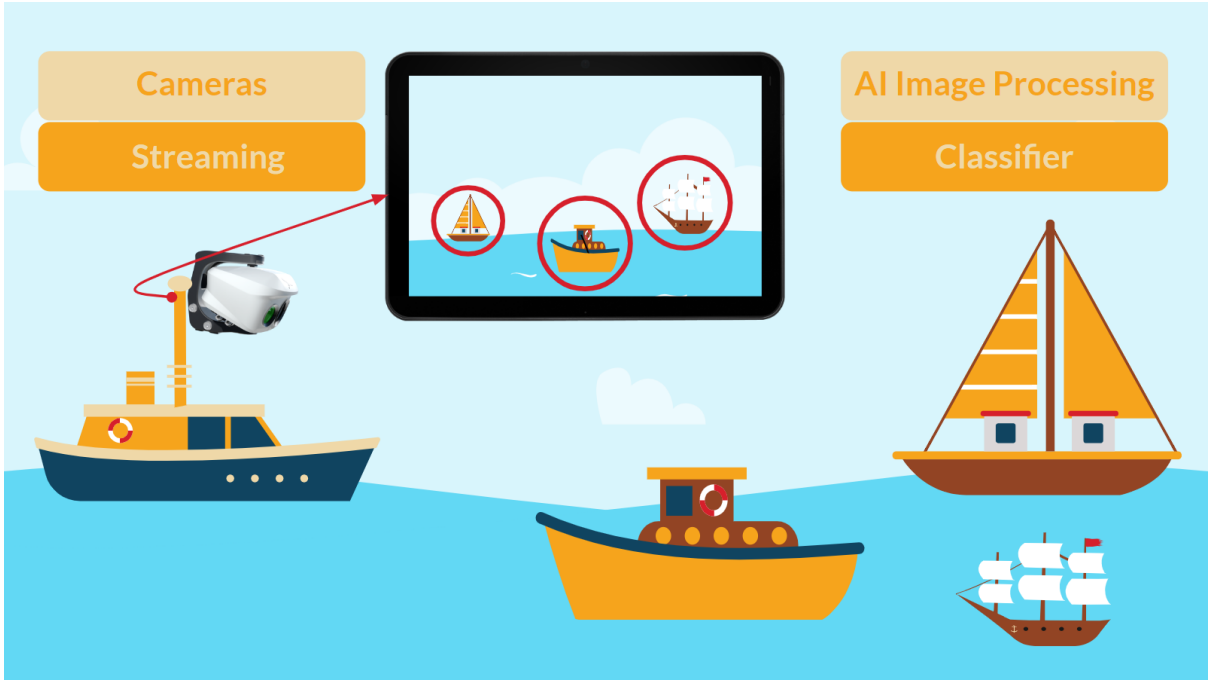

Fig. 3: Display of NaviDEC Work Context.

1.3.3 NaviDEC Use Case

Let's delve deeper into the operational intricacies. As previously mentioned, a boat is outfitted with multiple cameras, referred to as streaming, and an onboard AI image processor, known as a classifier.

The concept is to maintain continuous streaming on the boat. Regarding the AI image processing (classifier), given the modest computing capability of the deep edge on the boat, there is a need to either retain the mode or migrate the classifier element from the boat's deep edge to the land (referred to as edge) to leverage more powerful computing for AI image processing.

In the accompanying illustration below, a multi-interface router with transmission equipment, deployed on the boat by Sodira, establishes a wireless LTE connection to link the boat with the land-based Edge office through LTE connectivity.

- When the connection bandwidth between the boat and land is low, the AI image processing continues to operate from the boat, figure 4.
- As soon as the connection strengthens, the goal is to move the classifier component from the boat/deep edge to the land/Edge, allowing us to benefit from powerful computing for AI image processing while keeping streaming active on the boat, figure 5.
- Conversely, when the connection bandwidth diminishes, the action is taken to shift the classifier from land/Edge back to deep-edge/boat, figure 4.

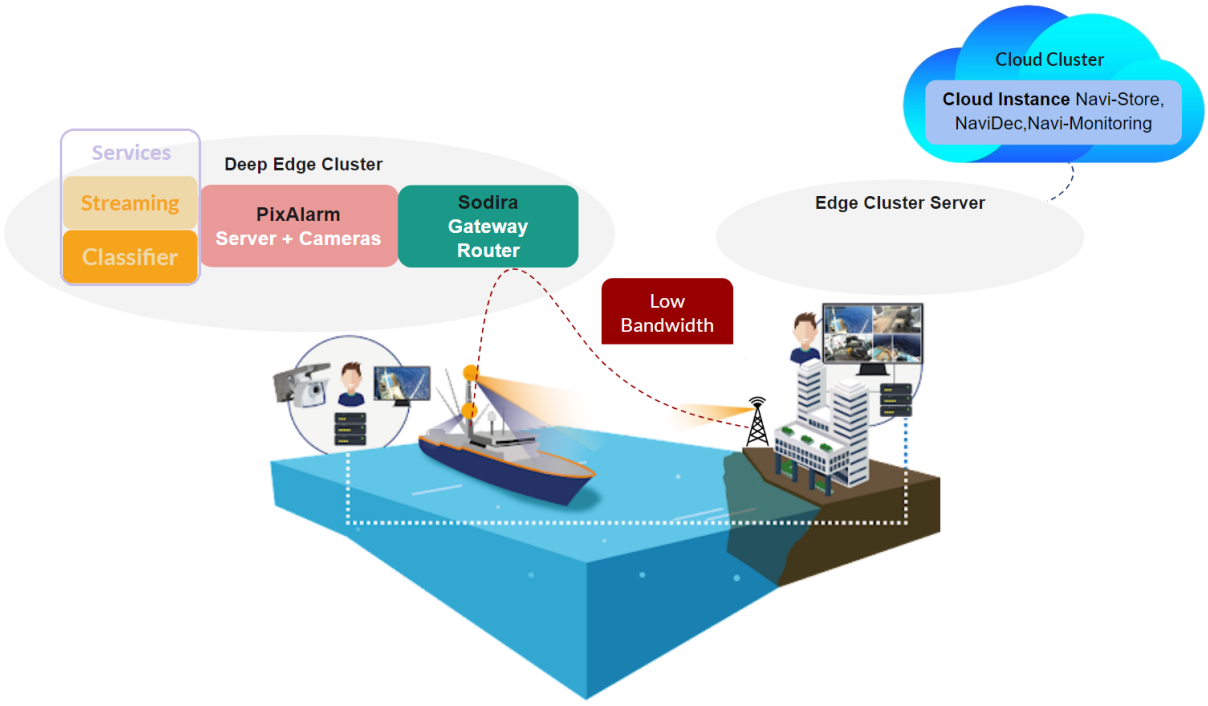

Fig. 4: The NaviDEC use-case is operating at low LTE bandwidth, streaming and classifier function on the boat.

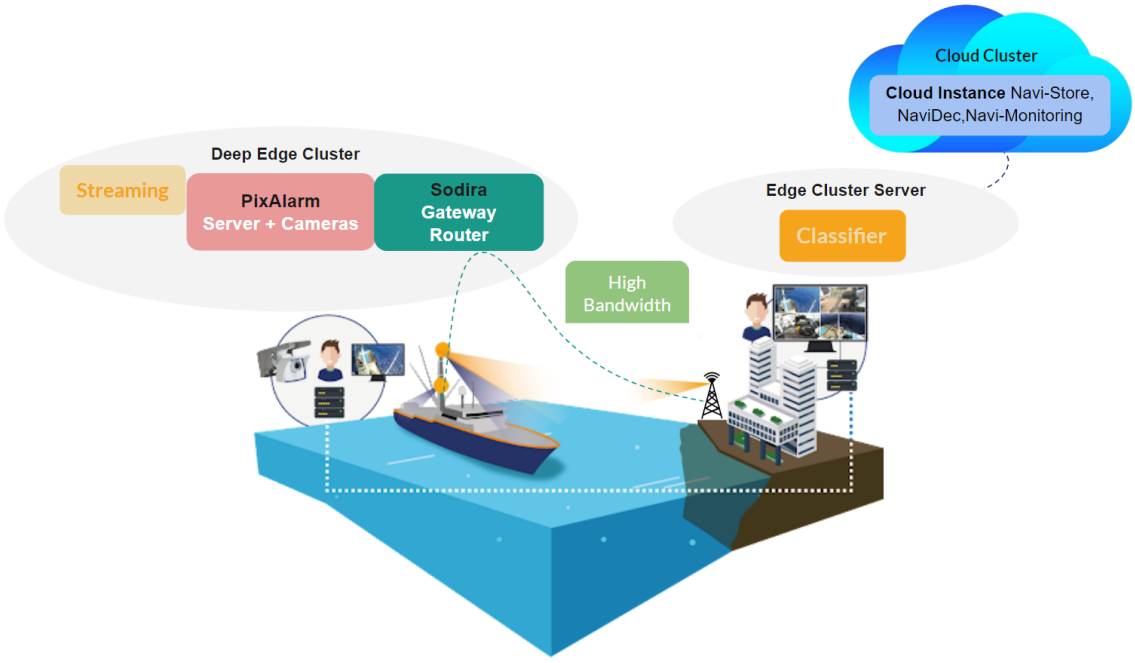

Fig. 5: The NaviDEC use-case is operating at High LTE bandwidth, only classifier transitions to Land/Edge.

1.3.4 IRISA Platform Architecture Overview

This section explores how the IRISA lab's Proof of Concept team demonstrates the platform's architecture. To set up the platform, one can connect to a remote Ubuntu v20 operating system from a laptop via an SSH terminal. Using Containernet for network emulation, two Docker containers and a RabbitMQ container for messaging are created. These containers, named "boat" and "land," represent the deep-edge and edge locations, respectively. Each container is equipped with Docker, a Minikube cluster, and Nginx for traffic handling.

Inside each Minikube cluster, the NaviDEC REST API is deployed, which governs the following behaviors:

1. Clients communicate with the REST API through HTTP requests.
2. The REST API translates these requests into kubectl API requests.
3. Responses are generated based on the outputs of the kubectl commands.
4. The master boat Minikube cluster communicates with the other master land Minikube cluster via HTTP for requests.

To manage this platform, four separate terminals are required:

1. The first terminal runs the NaviDEC REST API from the first Docker container in the master boat Minikube cluster.
2. The second terminal runs the NaviDEC REST API from the second Docker container in the master land Minikube cluster.
3. The third terminal operates the Meta orchestrator:
    - It is responsible for listing live clusters.
    - It makes decisions based on boat cluster prompts. For instance, when bandwidth improves, it provides the destination cluster's IP, enabling the boat cluster to move the classifier to cluster 2.
    - Communication between clusters and the Meta orchestrator is facilitated through RabbitMQ RPC.
4. The fourth terminal acts as a remote client, sending deployment and service requests to the boat cluster.

1.3.5 Principles of Operation for the IRISA Platform

Here are the key principles of operation for the IRISA Platform:

1. The client sends requests to the NaviDEC REST API in the master boat for the deployment and service deployment of streaming and classifier from the host 1 terminal.
2. The NaviDEC REST API in the master boat cluster interacts with the Kubernetes API, deploying streaming and classifier pods and services, figure 6.
3. When the boat cluster bandwidth improves, it requests the Meta orchestrator for the destination cluster to move the classifier. The Meta orchestrator provides the Edge cluster's IP address.
4. The boat cluster requests the Edge cluster to deploy the classifier resources. Once ready, the classifier is removed from the boat cluster, figure 7.
5. Conversely, when boat cluster bandwidth drops, the classifier is redeployed, and the Edge classifier is deleted.
6. Streaming always remains on the boat cluster.
7. Nginx on each cluster tailors streaming and classifier configurations based on its deployment location.

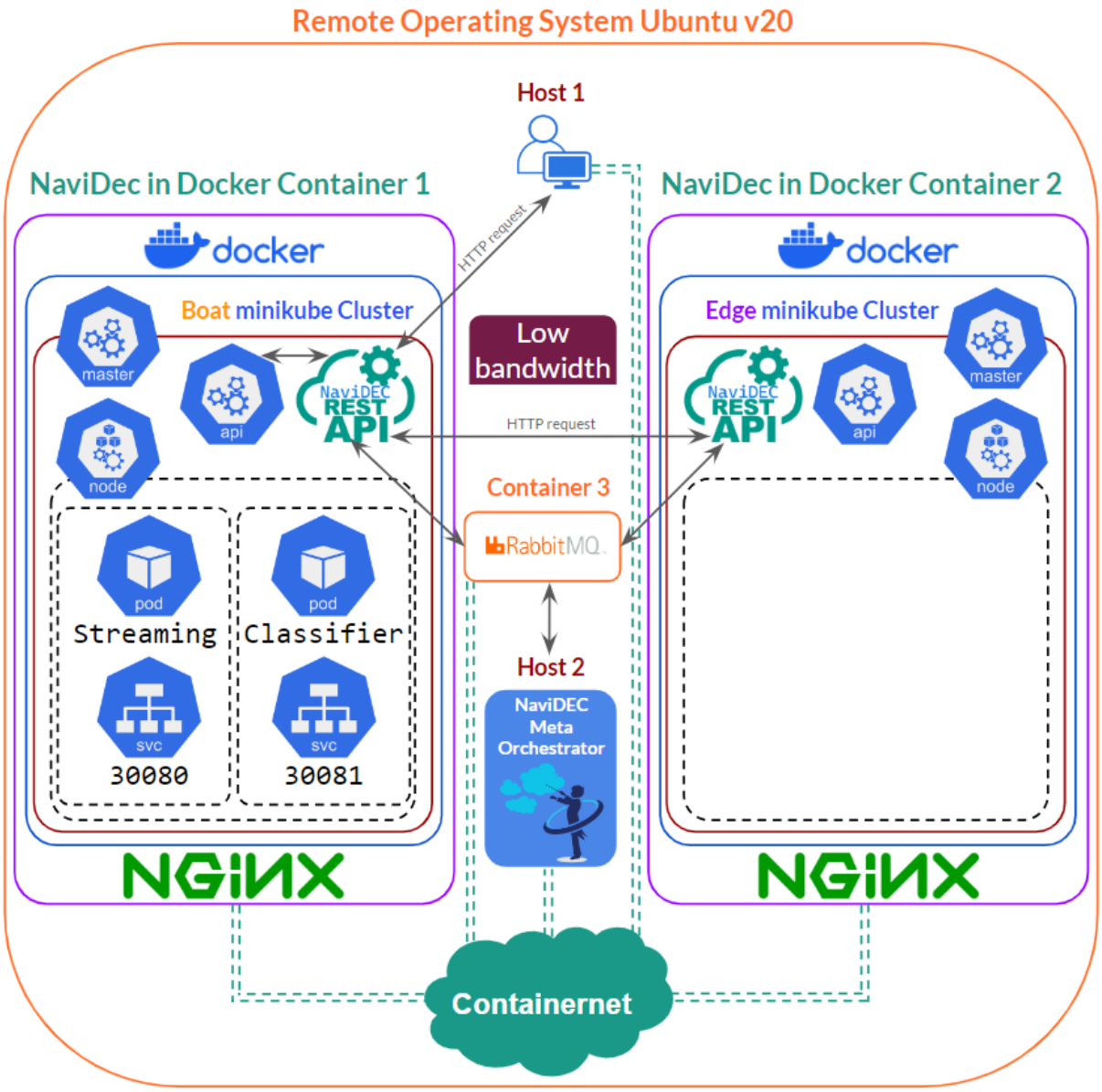

Fig. 6: IRISA platform is operating at a low bandwidth speed, streaming and classifier resources remain on boat.

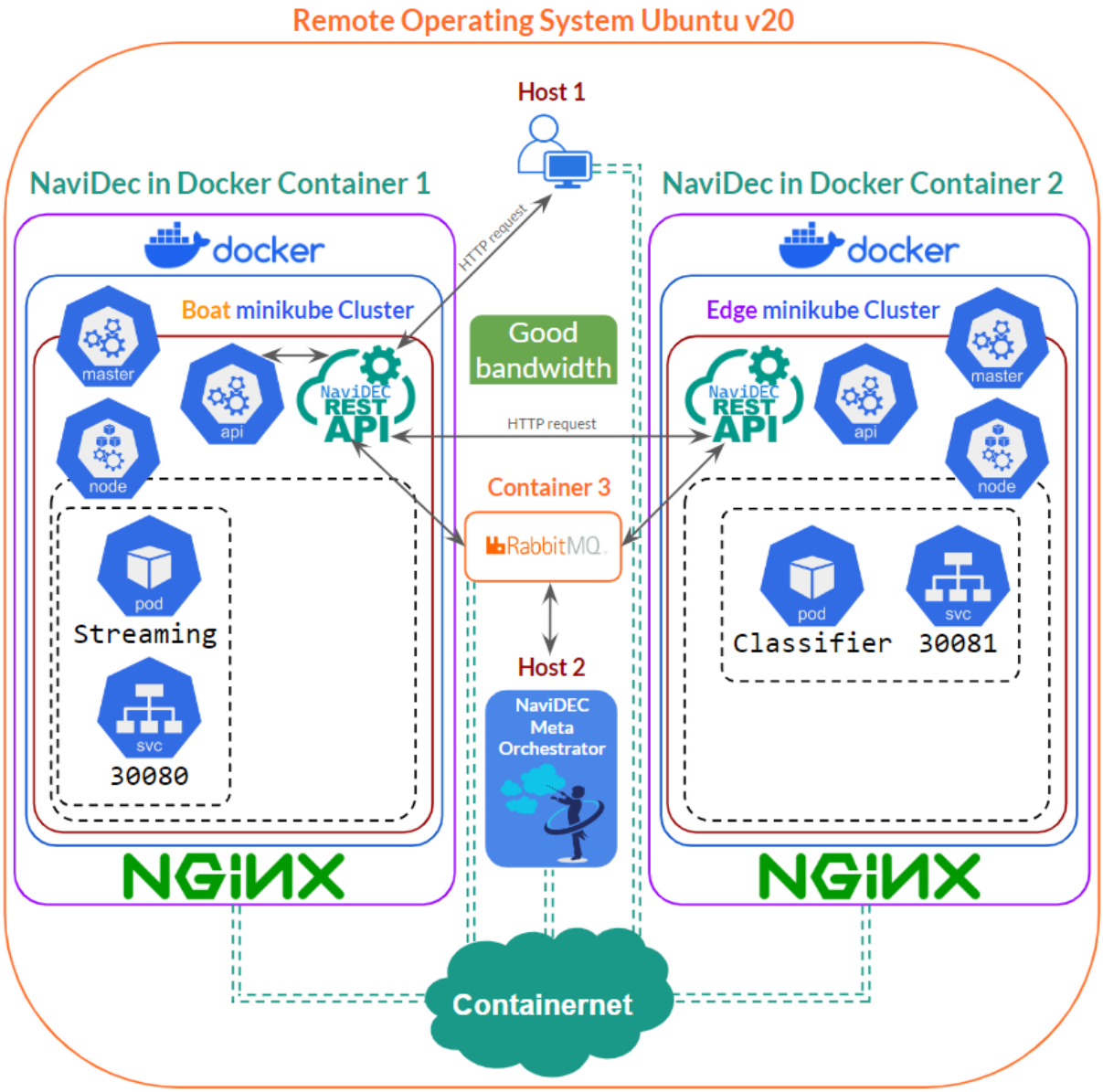

Fig. 7: IRISA platform is operating at a good bandwidth speed, only classifier resource are relocated to the Edge

1.3.6 Conclusion: Exploring the IRISA Platform Architecture

As we conclude our exploration of the IRISA platform architecture, we acknowledge its pivotal role in the project timeline. Born out of Phase One as a Proof of Concept, the IRISA platform sets the stage for the next phases of this innovative project. The initial design and setup, successfully carried out by a dedicated team member, provide a strong foundation for the current phase.

The complexities of the IRISA platform serve as a testament to the technical acumen involved and a bridge between theory and practice. By understanding its architecture, we gain insights into its workings, preparing us for the enhancements and refinements in Phase Two.

As we transition into the next chapter, we'll delve into my internship work during Phase Two. We'll detail the tasks executed, challenges overcome, and features enhanced, providing a comprehensive understanding of the platform's evolution. This narrative aligns with the project goal of advancing the platform, bringing us closer to the ultimate integration phase.

CHAPTER

TWO

IRISA PLATFORM ENHANCED CHARACTERISTICS

2.1 Introcuntion

As we embark on this chapter, we find ourselves in Phase Two of the NaviDEC project at IRISA. This phase, which is the focus of my internship, involves refining and enhancing the platform's features. Before diving into the specific enhancements I made to the platform, I will first describe how I inherited the project, along with several challenges, which I will outline in the following section titled 'Handover Highlights and Challenges'. An overview of the work carried out during Phase Two will then be provided. Following this overview, a detailed account of each task will be presented in the subsequent sections of this chapter, offering an in-depth look at the enhancements and improvements made to the platform. This will allow for a thorough understanding of the project's progression and the solutions implemented to overcome the challenges faced

As for how I organized myself over the course of five months to carry out this project, I will describe the methodology I used to organize my work in the 'Work Organization Methodology' section in the chapter that follows this one.

2.1.1 Handover Highlights and Challenges

The project, which served as a Proof of Concept (PoC) for the IRISA platform, was transitioned to my responsibility under the following circumstances:

- The transition of the Proof of Concept (PoC) IRISA platform project to me was a race against time. The engineer, who was my primary guide and the sole direct source of knowledge about the intricacies of the project's platform code, was set to leave the IRISA institution in just two weeks. This narrow window of time presented a steep learning curve. I had to absorb as much information and understanding about the project as I could, knowing that after this brief period, I would be navigating the complexities of the project on my own. Despite the time constraints, the engineer was incredibly supportive, going above and beyond to ensure I was well-informed about the entire project and patiently answering all my queries.
- The code for the Proof of Concept (PoC) IRISA platform project, which was handed over to me, contained deprecated elements and complex nested structures that added to the intricacy of the task at hand. This complexity necessitated a thorough understanding and subsequent refinement of the codebase.
- There was a lack of comprehensive documentation and flow charts, making it challenging to grasp the overall structure and functionality of the project.
- The architecture illustration provided was outdated, which meant that I had to invest time in understanding the current architecture and updating the illustration.
- The main concentration is on the Testbed platform that has been provided, which is convenient for both PoC and demonstration. On the other hand, the integration with related partners in a real-world project necessitates the transformation of the software version of the IRISA platform into a hardware version. This transformation process involves eliminating certain tools, such as the Containernet network emulator, and adjusting specific parameters to facilitate the integration process, among other things. This will be carried out in phase 3 of the project, after the internship period.

This section aims to paint a picture of the initial challenges I faced at the start of my internship. As an intern with limited experience, I found myself responsible for a complex, working platform that required a long list of enhancements. This responsibility was both daunting and exciting, and it pushed me to quickly learn and adapt.

In the following, I will list all accomplished tasks for the IRISA Platform. Then, in the subsequent sections during this chapter, I will detail each of the accomplished tasks with specifics. Finally, in the last chapter, I will elaborate on the methodology I utilized to organize my work and the tasks I completed for the platform. This will offer a comprehensive understanding of how I navigated these initial challenges and made significant progress in enhancing the platform.

2.1.2 Accomplished Tasks for IRISA Platform

In the following sections, we will delve into a detailed discussion of each task and its significance in the overall project. The tasks are categorized as follows:

1. **Mandatory Tasks:**

   These tasks formed the backbone of the project, encompassing the core responsibilities that were essential for the project's progression.

2. **Additional Tasks:**

   These tasks were undertaken in the intervals between supervisor meetings. They served to deepen my understanding of the project and significantly enhanced the quality of my work.

3. **Complementary Tasks:**

   While these tasks were not classified as mandatory or additional, they played a pivotal role in refining the project. Their contribution led to a more comprehensive and professional outcome.

The tasks spanned across various categories and included:

1. **Code Understanding:**

   This was a mandatory task that required a combination of tasks across all categories.

2. **Infrastructure Automation:**

   This additional task fell under the Platform Automation category.

3. **Dockerfile Build Code:**

   This was a complementary task within the Platform Automation category.

4. **Interactive Web-based Terminal:**

   This complementary task spanned across the Platform Automation, Development, Integration, and User Interface categories.

5. **Bandwidth Visualization with API Development:**

   This mandatory task involved the Platform Implementation, Automation, Development, Integration, and User Interface categories.

6. **Kubernetes Dashboard:**

   This mandatory task was part of the Platform Implementation, Automation, Development, Integration, and User Interface categories.

7. **Flask Python Framework:**

   This complementary task was part of the Platform Migration and Development categories.

8. **Comprehensive Logging System:**

   This complementary task was part of the Platform Documentation and Development categories.

9. **Flowchart Diagram Illustration:**

   This additional task was part of the Platform Documentation and Development categories.

10. **Online Documentation:**

    This mandatory task was part of the Platform Documentation, Development, and User Interface categories.

11. **Automated Build and Deployment Pipeline in GitLab:**

    This complementary task was part of the Platform Automation category.

12. **OpenVPN Integration:**

    This complementary task was part of the Platform Automation and Integration categories.

13. **RPC Server Web Browser Launcher Tool:**

    This complementary task was part of the Platform Implementation, Automation, Development, Integration, and User Interface categories.

As we can see, a total of thirteen tasks were successfully accomplished: four were mandatory, two were additional, and seven were complementary. Regardless of their classification, each task played a crucial role in the successful execution and enhancement of the project.

2.2 Code Understanding

2.2.1 Introduction

Understanding the code was a progressive journey, unfolding with each task. Each completed task clarified the code's intricacies. The final task provided a comprehensive view of the codebase. Subsequent tasks, listed in an order that gradually contributed to code clarity, unlocked a clearer understanding with each step. Considering all tasks represents the journey in understanding the codebase.

2.2.2 Structure of the Following Sections

The subsequent sections in this chapter are each dedicated to a distinct task that has been completed. This method enhances clarity, and we will adhere to this structure wherever feasible.

Example Task: *RPC SERVER WEB BROWSER LAUNCHER*

The structure, accompanied by explanations, is organized as follows whenever feasible:

1. **Introduction: Revolutionizing Remote Control with RPC Server Web Browser Launcher**
   - A concise introduction to the task.
2. **Why RPC Server Web Browser Launcher within IRISA Platform**
   - An explanation of why this task is important.
3. **Benefits of Integrating RPC Server Web Browser Launcher with IRISA Platform**
   - The benefits derived from this integration.
4. **In-Depth Examination of RPC Server Web Browser Launcher of IRISA Platform**
   - A simplified breakdown of the code, avoiding technical jargon as much as possible.
5. **Illustration: RPC Server Web Browser Launcher**
   - Present the pertinent image, where applicable.
6. **In summary**
   - Recap of the role that the intended tool plays in IRISA platform, where applicable.
7. **Code Repository**
   - Provide the link to the specific file or folder that substantiates the explained task, where applicable.

2.3 Ansible Playbook, An Infrastructure Automation Tool

2.3.1 Introduction: Leveraging Ansible for Infrastructure Orchestration

In the dynamic landscape of modern infrastructure management, the IRISA platform harnesses the power of Ansible, a robust open-source automation tool, to streamline and automate server configuration. Ansible's versatility and simplicity make it an ideal choice for orchestrating complex tasks, offering a range of benefits that significantly contribute to the efficiency and reliability of the IRISA Platform.

2.3.2 Why Ansible?

At the core of the IRISA platform's infrastructure automation lies the Ansible playbook *configure_server.yml*. Ansible, known for its agentless architecture and YAML-based syntax, provides a declarative language for describing system configurations, making it an excellent fit for platform. The choice of Ansible aligns with the project's commitment to simplicity, flexibility, and scalability,, as highlighted by the following advantages when compared with other tools:

1. **Simplicity and Ease of Use:**

   Ansible is known for its simplicity and ease of use. It uses YAML-based "playbooks" to configure systems, deploy software, and orchestrate advanced workflows[1]. This makes it easier to read and write compared to the domain-specific languages used by Puppet and Chef[2].

2. **Agentless Architecture:**

   Unlike Puppet, Chef, and Salt, Ansible does not require the installation of an agent on managed nodes. This simplifies the deployment process and reduces the resources required on the servers you are managing.

3. **Flexibility:**

   Ansible has great flexibility in the types of workloads it can effectively run. It is suited to critical production environments where there is a greater need for granular control[3].

4. **Strong Support for Cloud-Based Infrastructure:**

   Ansible has robust support for cloud-based infrastructure[1], making it a good choice if you're working with cloud platforms.

5. **Imperative Programming Paradigm:**

   Ansible allows users to script commands in an imperative programming paradigm[4], which can be more intuitive for users who are used to scripting and procedural programming[2].

6. **Community and Enterprise Support:**

   Ansible has a large community of global contributors and is backed by Red Hat, which offers enterprise support[1].

[1] Understanding Ansible, Terraform, Puppet, Chef, and Salt. https://www.redhat.com/en/topics/automation/understanding-ansible-vs-terraform-puppet-chef-and-salt.

[2] Why we use Terraform and not Chef, Puppet, Ansible, Pulumi, or ... - Medium. https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c.

[3] Chef vs. Puppet vs. Ansible vs. SaltStack - Our Comparison. https://www.justaftermidnight247.com/insights/chef-vs-puppet-vs-ansible-vs-saltstack-configuration-management-tools-compared/.

[4] Puppet vs. Chef vs Ansible vs SaltStack | JetPatch. https://jetpatch.com/blog/agent-management/puppet-vs-chef-vs-ansible-vs-saltstack/.

2.3.3 Benefits of Ansible within IRISA Platform

Here are some benefits of using Ansible within the IRISA platform:

1. **Consistency and Reproducibility:**

   Ansible ensures consistent server configurations across different environments. By defining tasks in a playbook, the IRISA team can reproduce the same infrastructure settings reliably, minimizing the risk of configuration drift and ensuring consistency in deployments.

2. **Efficiency in Configuration Management:**

   The *configure_server.yml* playbook encapsulates a series of tasks, ranging from system checks to software installations. Ansible's idempotent nature ensures that running the playbook multiple times does not cause unintended changes, providing efficiency in configuration management without unnecessary modifications.

3. **Cross-Platform Compatibility:**

   Ansible's agnostic approach to underlying infrastructure allows seamless management of heterogeneous environments. Whether the target is a local development machine or a remote server, Ansible abstracts away the complexity, allowing the IRISA team to focus on tasks rather than platform intricacies.

4. **Collaborative Development:**

   The playbook serves as a collaborative artifact, capturing the expertise of the IRISA team in a human-readable format. Ansible playbooks are version-controlled, facilitating teamwork, code reviews, and knowledge sharing among team members, contributing to a transparent and collaborative development process.

5. **Extensibility and Integration:**

   Ansible's extensive library of modules and integrations simplifies the integration of diverse technologies within the Platform. From SSH session configurations to the installation of Docker, Containernet, and various packages, Ansible empowers the project to integrate seamlessly with a wide array of tools.

6. **Auditability and Documentation:**

   The declarative nature of Ansible playbooks enhances auditability. Each task within *configure_server.yml* serves as self-documenting code, making it easier for team members to understand, modify, and maintain the infrastructure over time. This contributes to a robust documentation culture within the Platform.

2.3.4 In-Depth Examination of Ansible Playbook Operations of IRISA Platform

The *configure_server.yml* playbook is crafted to set up the server intended for use in the IRISA platform. It undertakes the following tasks:

- **Configuration for localhost**
  - Specify target hosts as localhost.
  - Gather facts about the system.
  - Execute tasks with elevated privileges.
- **Checking Ubuntu Version**
  - Ensure the target system is running Ubuntu 20.xx.
- **Set SSH Session Timeout Settings**
  - Set TCPKeepAlive, ClientAliveInterval, ClientAliveCountMax in SSH configuration.
  - Restart the SSH service.
- **Add NOPASSWD Rule to sudoers**
  - Add a NOPASSWD rule to the sudoers file.
  - Print success or error messages.

- **Setup Current Directory and Destination Directory**
  - Get the current directory.
  - Set the destination directory based on the current directory.
- **Copy run.sh from navidec to parent directory**
  - Copy the *run.sh* file from *navidec* to the parent directory.
  - Change the owner of *run.sh*.
- **Set Date, Time, and Timezone**
  - Ensure NTP is installed.
  - Set the system timezone to Europe/Paris.
  - Synchronize system time with an NTP server.
  - Display the current date and time.
- **Install OpenVPN**
  - Update packages and install required packages.
  - Add OpenVPN repository key and repository.
  - Install OpenVPN.
  - Configure OpenVPN settings.
  - Restart OpenVPN Access Server.
- **Install Containernet**
  - Check if Containernet repository already exists.
  - Remove existing Containernet repository if it exists.
  - Clone Containernet repository.
  - Run Containernet install playbook.
- **Update package list and Install packages**
  - Update the package list.
  - Install packages: sshpass, latexmk, texlive-fonts-recommended, texlive-latex-extra, texlive-fonts-extra.
- **Install Kubectl**
  - Download and install Kubectl.
  - Remove the Kubectl setup file.
- **Install Sysbox**
  - Download and install Sysbox.
  - Install jq.
  - Remove Sysbox setup file.
  - Check Sysbox service status.
  - Start Sysbox service if inactive.
- **Update Docker daemon and restart it**
  - Insert a line into the Docker daemon configuration.
  - Restart the Docker daemon.
- **Install packages**
  - Install Python packages using pip3.

- **Install Node.js and nodemon**
  - Remove existing Node.js and NodeSource repository configurations.
  - Remove NodeSource repository list file.
  - Remove NodeSource GPG key.
  - Update and install required packages.
  - Download NodeSource GPG key.
  - Save GPG key to /etc/apt/keyrings/nodesource.gpg.
  - Add NodeSource to sources list.
  - Update and install Node.js.
  - Check Node.js version.
  - Print Node.js version.
  - Upgrade npm using sudo.
  - Print npm upgrade result.
  - Install nodemon using sudo or npm.
  - Print nodemon installation result.
- **Export COLORTERM=truecolor to current user**
  - Get the current user.
  - Determine the user's home directory.
  - Check and add export COLORTERM=truecolor to ~/.bashrc if not present.
- **Create docker group & add current user to it**
  - Get the current non-root remote user.
  - Add docker group.
  - Add the user to the docker group.

The playbook covers a range of configurations, from system settings to the installation of various tools and applications, aimed at setting up a server environment for development and containerization.

2.3.5 Playbook Usage

The playbook can be run using the following command:

Before deploying IRISA Platform, it is a prerequisite to prepare the server. The server should be running Ubuntu 20.04 LTS.

First, establish an SSH connection to the remote Ubuntu server and initiate the installation of Ansible on a freshly installed version of Ubuntu 20.xx:

```
sudo apt-get update && sudo apt-get install ansible -y
```

Next, clone the NaviDec repository:

```
git clone -b master https://gitlab.inria.fr/muhammad.al-qabbani/navidec.git
```

Lastly, run the Ansible playbook to install the required dependencies:

```
sudo ansible-playbook -i "localhost," -c local navidec/ansible/configure_server.yml &&
↪ exit
```

After the installation is complete, the current active terminal session will automatically log out. Logging back in will ensure that the Docker group membership is re-evaluated.

Note: It is assumed that Ansible is installed on the system for this playbook.

Warning: Be cautious when making changes to system configurations, especially when using *become* and *sudo* privileges.

2.3.6 In Summary

In essence, the incorporation of Ansible in the IRISA Platform is a strategic decision aimed at achieving not just automation but a sustainable, scalable, and collaborative approach to infrastructure management. As we delve into the details of the *configure_server.yml* playbook, the power of Ansible unfolds, showcasing its role in shaping the robust and efficient infrastructure that underpins the IRISA platform.

2.3.7 Code Repository

Please refer to the *configure_server.yml* playbook source code for detailed task configurationsm[5][6].

For more details, Click here to view the playbook source code.

[5] For more information on Ansible, refer to the official documentation: https://docs.ansible.com/
[6] For information on YAML syntax, refer to the YAML documentation: https://yaml.org/

2.4 Docker Container Automation

2.4.1 Introduction: Embracing Docker for Containerized Efficiency

In the ever-evolving landscape of modern software development, the IRISA platform strategically adopts Docker, an industry-standard containerization tool, to elevate the efficiency and consistency of its container orchestration. Docker's lightweight, portable containers offer a versatile solution, providing a multitude of benefits that significantly enhance the deployment and scalability aspects of the IRISA Platform.

2.4.2 Why Docker?

At the heart of the IRISA platform's containerization strategy lies the Dockerfile *Dockerfile.cnet_nind*. Docker, renowned for its containerization approach and user-friendly workflows, emerges as the ideal choice for encapsulating applications and dependencies within isolated environments. This choice aligns seamlessly with the IRISA Platform's pursuit of scalability, consistency, and seamless deployment.

Docker offers several advantages over traditional virtual machines (VMs):

1. **Efficiency:**

   Docker containers are lightweight and share the host system's kernel, making them more resource-efficient than VMs. This allows you to run more Docker containers on a given hardware combination than VMs[7].

2. **Portability:**

   Docker containers can run anywhere, on any machine that has Docker installed, without worrying about the underlying system. This makes it easy to develop on one machine, test on another, and deploy on a third, all without any hiccups[Page 24, 7].

3. **Speed:**

   Docker containers are faster to launch than VMs because they don't need to boot up a full operating system[8].

4. **Docker Hub:**

   Docker has a large repository of images available from Docker Hub, which can be used as a starting point for your own applications[9].

5. **Broad Functionality:**

   Docker is a comprehensive platform that includes a container runtime, image build tools, a registry for hosting container images, and an orchestration engine. This broad functionality makes Docker a one-stop solution for many developers[9].

6. **Large Community and Ecosystem:**

   Docker has a large community of users and developers, which means you can find a wealth of resources, tutorials, and pre-built images to help you get started[9].

[7] Docker vs Virtual Machine (VM) Key Differences You Should Know. https://www.freecodecamp.org/news/docker-vs-vm-key-differences-you-should-know/.

[8] Docker vs Virtual Machine: Where are the differences?. https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/.

[9] Docker Alternatives to Look Out for in 2023 | JFrog. https://jfrog.com/devops-tools/article/alternatives-to-docker/.

2.4.3 Benefits of Using Docker within IRISA Platform

Here are some benefits of using Docker within the IRISA platform:

1. **Isolation and Portability:**

   Docker containers encapsulate applications and their dependencies, ensuring isolation from the underlying infrastructure. This isolation guarantees consistent behavior across various environments, facilitating portability and eliminating the notorious "it works on my machine" issue.

2. **Efficient Resource Utilization:**

   Docker's lightweight containers share the host OS kernel, optimizing resource utilization. This efficiency translates into faster start-up times, reduced overhead, and the ability to run multiple containers on a single host without compromising performance.

3. **Streamlined Development Workflow:**

   The Dockerfile *Dockerfile.cnet_nind* serves as a declarative script for defining the container's environment. This script simplifies the development workflow, allowing the IRISA team to codify dependencies, configurations, and application setup in a consistent and reproducible manner.

4. **Consistent Deployments Across Environments:**

   Docker ensures consistency between development, testing, and production environments. The containerized application, defined by the Dockerfile, guarantees that what is developed and tested locally mirrors the production environment, reducing deployment-related surprises.

5. **Scalability and Microservices Architecture:**

   Docker facilitates the adoption of a microservices architecture, allowing the IRISA Platform to break down monolithic applications into smaller, manageable components. This scalability enables more straightforward maintenance, updates, and a modular approach to development.

6. **Version Control and Image Repositories:**

   Docker introduces version control to containerized applications through images. The Dockerfile, when combined with version control systems, enables the IRISA team to track changes systematically. Docker Hub or other container registries store and distribute these versioned images, facilitating collaboration and seamless deployment.

7. **Fast and Reproducible Builds:**

   The Dockerfile automates the build process, ensuring fast and reproducible builds. This accelerates the development lifecycle, allowing the IRISA team to iterate quickly, test efficiently, and deploy with confidence.

8. **Simplified Dependency Management:**

   The Dockerfile explicitly defines dependencies and their configurations. This not only simplifies the installation process but also mitigates potential conflicts, ensuring that each containerized instance runs with precisely the required dependencies.

In essence, the integration of Docker within the IRISA Platform is a strategic choice aimed at optimizing the development, deployment, and scalability aspects. The Dockerfile *Dockerfile.cnet_nind* stands as a testament to the power of Docker, showcasing its role in crafting a streamlined and efficient container orchestration environment for the IRISA platform. This document provides a comprehensive overview of the Dockerfile, delving into its intricacies and highlighting the transformative benefits it brings to the IRISA Platform.

2.4.4 In-Depth Examination of the Dockerfile.cnet_nind Operations within IRISA Platform

The *Dockerfile.cnet_nind* serves as a blueprint for creating a Docker image tailored to the infrastructure needs of the IRISA Platform. It orchestrates the installation of essential tools, configurations, and dependencies within a Docker container. Let's break down the content of the Dockerfile:

- **Base Image:**

  Specifies the base image as *ubuntu:22.04*, laying the foundation for subsequent installations and configurations.

- **Setting Timezone:**

  Sets the timezone within the container to Europe/Paris (*ENV TZ=Europe/Paris*), ensuring consistent time settings.

- **Installing Basic Tools:**

  Updates the package list and installs various tools, including curl, iputils-ping, apt-utils, lsof, sudo, and others, to establish a robust development environment.

- **Python and Pip Installation:**

  Installs Python 3 and pip, ensuring the availability of essential Python packages for subsequent tasks.

- **Docker Installation:**

  Downloads and installs Docker using the official installation script, equipping the container with Docker capabilities for managing and running containers.

- **Minikube Installation:**

  Installs Minikube, a tool for running Kubernetes clusters locally, enhancing the container's capability to simulate Kubernetes environments.

- **Kubectl Installation:**

  Installs Kubectl, the Kubernetes command-line tool, facilitating interaction with Kubernetes clusters.

- **Helm Installation:**

  Installs Helm, the Kubernetes package manager, using a script, providing a mechanism for deploying and managing applications on Kubernetes.

- **Nginx Installation:**

  Installs Nginx, the popular web server, to support web server functionality within the container.

- **Cleaning Up:**

  Executes cleanup commands, removing unnecessary packages and optimizing the size of the final Docker image.

- **Entrypoint Script:**

  Copies an entrypoint script (*startup-nind.sh*) into the container. This script contains instructions to be executed when the container starts.

2.4.5 In-Depth Examination of Entrypoint Script Operations within IRISA Platform

The *startup-nind.sh* script, designated as the entry point for the Docker container, plays a crucial role in orchestrating various tasks upon container initialization. Let's delve into its functionalities:

- **Command-Line Argument Parsing:**

  The system parses command-line arguments, specifically looking for the --delete-minikube flag. If this flag is detected, it initiates the deletion of the Minikube cluster. Notably, this flag is utilized automatically during the Docker image creation process as a cleanup mechanism.

- **Docker API Accessibility Check:**

  Ensures the Docker daemon is running and restarts it if necessary. If the Docker API is not accessible, the script restarts the Docker daemon, guaranteeing a stable environment for subsequent operations.

- **Minikube Startup:**

  Initiates the Minikube cluster, verifying the availability of the Docker daemon. If Minikube fails to start, the script retries the operation after deleting the existing Minikube cluster.

- **Deleting Minikube Cluster:**

  If the *--delete-minikube* flag is set, the script deletes the Minikube cluster and exits gracefully.

- **Starting Nginx:**

  Starts the Nginx web server, facilitating web server functionality within the container.

- **Continuous Docker Daemon Monitoring:**

  Enters a loop, periodically invoking the *check_docker* function to ensure the Docker daemon remains running throughout the container's lifecycle.

2.4.6 In summary

The *Dockerfile.cnet_nind* orchestrates the construction of a Docker image with a comprehensive set of tools and configurations. From foundational utilities to advanced technologies like Docker and Kubernetes, the Dockerfile ensures that the resulting container is well-equipped for development, testing, and container orchestration within the IRISA Platform's ecosystem.

startup-nind.sh serves as the orchestrator for initializing the Docker container. It checks and ensures the availability of the Docker daemon, starts Minikube with necessary configurations, and commences Nginx for web server functionality. The script exhibits a dynamic quality by continuously monitoring and restarting the Docker daemon when needed, ensuring a resilient and stable containerized environment.

This script aligns seamlessly with the overarching purpose of the *Dockerfile.cnet_nind*, contributing to the comprehensive set of tools and configurations that make the resulting Docker image well-suited for development, testing, and container orchestration within the IRISA Platform's ecosystem.

2.4.7 Code Repository

For more details, Click here to view the Dockerfile.cnet_nind source code.

For more details, Click here to view the startup-nind.sh source code.

2.5 Interactive Web-based Terminal

2.5.1 Introduction: Harnessing Power of xterm.js for Interactive Web Terminals

The IRISA Platform presents an innovative web-based terminal system, developed in Node.js and powered by xterm.js. This system consolidates four terminals into a single, user-friendly page, ensuring a responsive design and automated execution. The interactive terminal allows command execution, process monitoring, and interaction with the platform environment, all from one place, which enhances productivity and convenience.

2.5.2 Why xterm.js?

Initially, the platform used Xterm, a terminal emulator for the X Window System, which provides a command-line interface for users to interact with the system. Xterm is typically used on Unix-like operating systems. However, when four Xterm windows are run, they pop up from the SSH terminal in an unorganized manner, making it difficult to align and track them each time the platform runs. Furthermore, executing commands manually in each window adds to the burden and is time-consuming.

To address the need for organizing four terminals inside one window, running simultaneously and with automated execution, Xterm.js was implemented. Xterm.js is a JavaScript library that enables the building of terminal applications that run in a web browser.

Here are some of its key features:

1. **Terminal apps compatibility:**

   Xterm.js works with most terminal apps such as bash, vim, and tmux, including support for curses-based apps and mouse events[10],[11].

2. **Performance:**

   Xterm.js is really fast and even includes a GPU-accelerated renderer[Page 28, 10],[11].

3. **Rich Unicode support:**

   It supports CJK, emojis, and IMEs[10],[11].

4. **Self-contained:**

   Xterm.js requires zero dependencies to work[10],[11].

It's used in several world-class applications to provide great terminal experiences, such as SourceLair, an in-browser IDE, and Microsoft Visual Studio Code, a modern, versatile, and powerful open-source code editor[12].

2.5.3 Benefits of xterm.js within IRISA Platform

Here are some benefits of using xterm.js within the IRISA platform:

1. **Responsive Web Design (RWD):**

   xterm.js ensures a responsive user interface, adapting to various screen sizes and resolutions. This responsiveness enhances the user experience, allowing for optimal terminal interaction across different devices.

2. **Integration with Python and Node.js:**

   The integration of xterm.js with both Python and Node.js showcases its flexibility in accommodating diverse technologies. This compatibility facilitates a seamless connection between server-side logic (Python) and client-side interactivity (JavaScript).

[10] GitHub - xtermjs/xterm.js: A terminal for the web. https://github.com/xtermjs/xterm.js/.
[11] xterm - npm. https://www.npmjs.com/package/xterm.
[12] Xterm.js. https://xtermjs.org/.

3. **Four Terminals in a Single Page:**

   NaviDEC leverages xterm.js to organize and manage four terminals efficiently within a single web page. This organization enhances the user's ability to work with multiple instances, promoting a streamlined and organized terminal experience.

4. **Fully Automated Execution:**

   Xterm.js contributes to the IRISA Platform's efficiency by enabling fully automated executions. This ensures that complex tasks, configurations, and interactions within the terminals are handled effortlessly, minimizing manual intervention.

2.5.4 In-Depth Examination of xterm.js Integration within IRISA Platform

The xterm.js integration involves configuring and managing four terminals within a web page. The server.js file orchestrates this integration, utilizing express, socket.io, and node-pty to create, control, and communicate with the terminals. The accompanying index.html file establishes the structure for rendering the terminals within the browser.

By combining xterm.js with NaviDEC's Python and Node.js subprojects, the project achieves a sophisticated yet user-friendly solution for interactive web-based terminals. The seamless integration of xterm.js aligns with the IRISA team's commitment to providing an efficient and powerful platform.

Implementation Details:

The subproject is implemented using Node.js, Express, and the xterm.js library. The *server.js* file acts as the backend server, facilitating communication between the client and the terminals. Each terminal instance is created using the xterm.js library and is connected to the server through Socket.IO.

Terminal Organization:

The terminals are organized within a flex container, allowing for a responsive and visually appealing layout. Each terminal container includes a header displaying the terminal's purpose, such as "Terminal 1 (Boat)," providing users with clear context for each terminal's role.

Terminal Initialization:

Upon connection to the server, each terminal is initialized with specific commands relevant to the IRISA Platform. This includes tasks such as starting services, executing scripts, and configuring network elements.

User Interaction:

Users can interact with the terminals by entering commands, and the output is dynamically displayed in the corresponding terminal window. Additionally, the web-based terminal interface supports resizing, adapting to changes in the browser window size.

2.5.5 Illustration: Embedding four Xterm.js Terminals into IRISA Platform Web Browsers

Here are three figures comparing and illustrating the previous layout of four Xterm terminals with the enhanced version, showcasing the integration of Xterm.js into web browsers on the IRISA platform.

Previously, on an outdated feature on platform, the display of four Xterm terminals used to appear as scattered pop-ups from an SSH terminal. However, recent enhancements have rendered this feature obsolete.

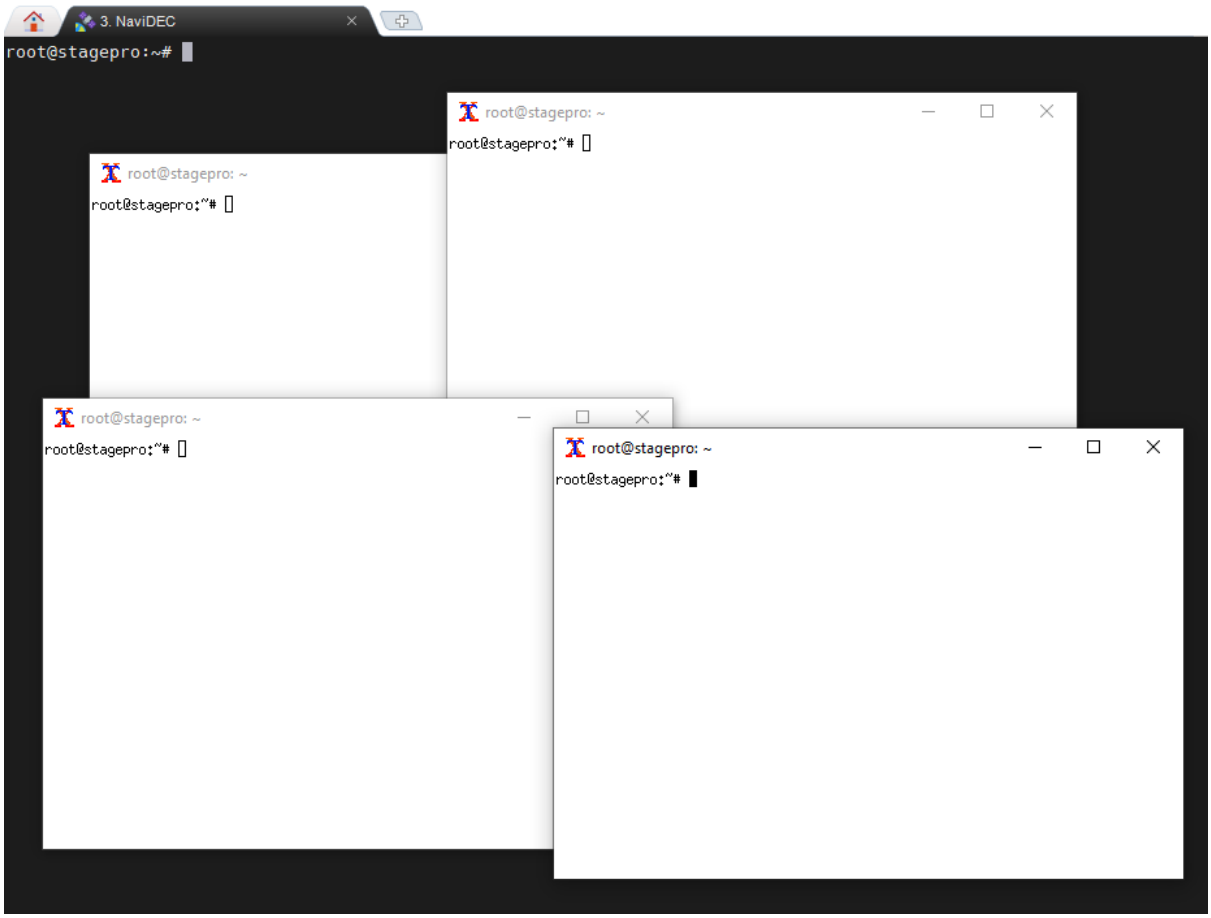

Fig. 1: Visualization of the outdated and deprecated feature: Four Xterm Terminals displayed as scattered pop-ups originating from an SSH Terminal.

The next illustration presents a web browser with an open tab directed to the server name, followed by its port address, specifically, http://stagepro.tk:3001. Within this browser, four interactive web terminals, powered by xterm.js, are displayed:

- Terminal 1, located at the upper left, represents the Boat.
- Terminal 2, situated at the upper right, signifies the Edge.
- Terminal 3, found at the lower left, represents the first instance of the Host.
- Terminal 4, located at the lower right, represents the second instance of the Host.

When the browser is automatically opened, this tab activates, triggering a series of automated commands on each terminal:

- Terminal 1 initiates the NaviDEC REST API.
- Terminal 2 also runs the NaviDEC REST API.

- Terminal 3 sends an automated request to the NaviDEC REST API on Terminal 1 (the Boat), deploying the streaming and classifier resources. It then begins monitoring the current streaming bandwidth, which is currently indicated as 'BAD' due to low speed.
- Terminal 4 runs the Meta Orchestrator via an automated command, listing all available terminals. Here, 'cluster1' for the Boat and 'cluster2' for the Edge are visible.

Under low-speed conditions, Terminal 1 displays the streaming and classifier deployments along with their services. In contrast, Terminal 2, representing the Edge, shows no deployed resources.

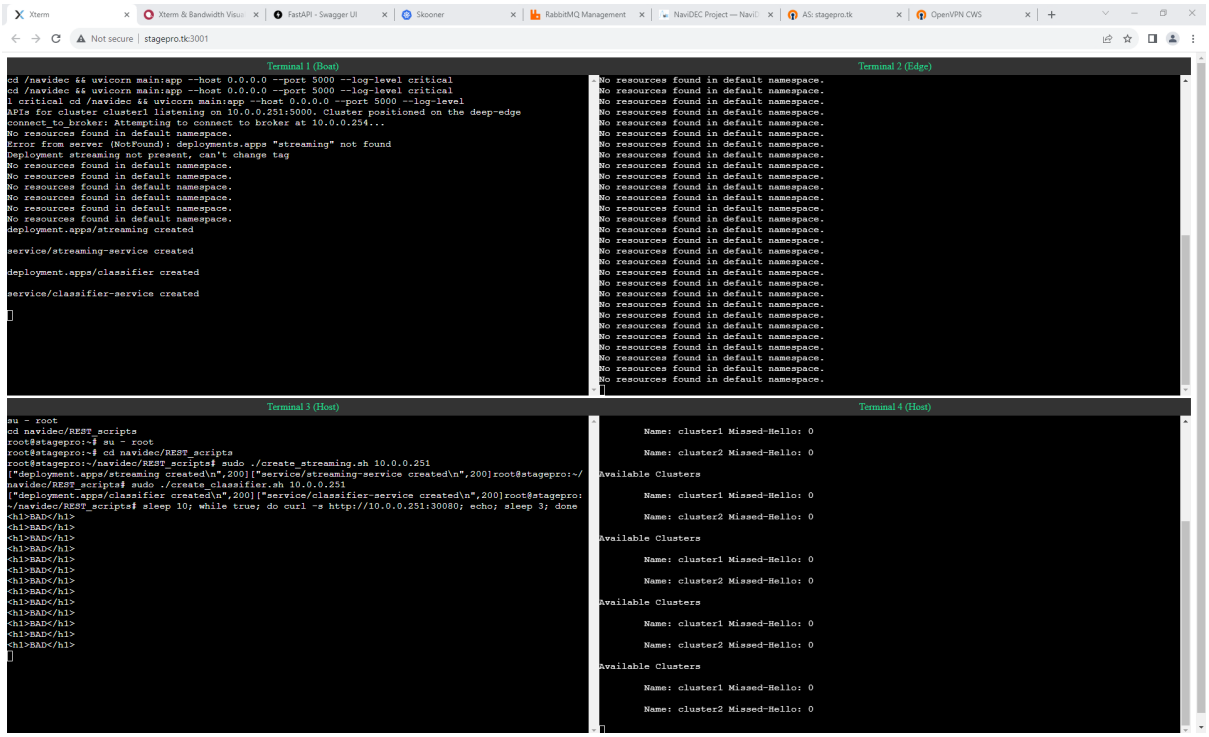

Fig. 2: Display of Four Xterm.js Terminals Embedded in Web Browsers Under Low Bandwidth Condition.

Now, since the bandwidth has improved to high, as indicated by the 'GOOD' status in Terminal 3 of the Host, we can observe changes in the resources:

- The streaming deployment tag in cluster1 on Terminal 1 (Boat) has changed to 'Good'. then, the classifier deployment resource has been deployed in cluster2 on Terminal 2 (Edge) and removed from cluster1.
- In Terminal 1, we can see the 'AbroadResources' that contain the classifier service and deployment.
- In Terminal 2, within cluster2, the 'ForeignResources' are shown for the classifier service and deployment.
- Terminal 4 (Host) continues to update the available clusters, and we can see the list for cluster1 and cluster2.

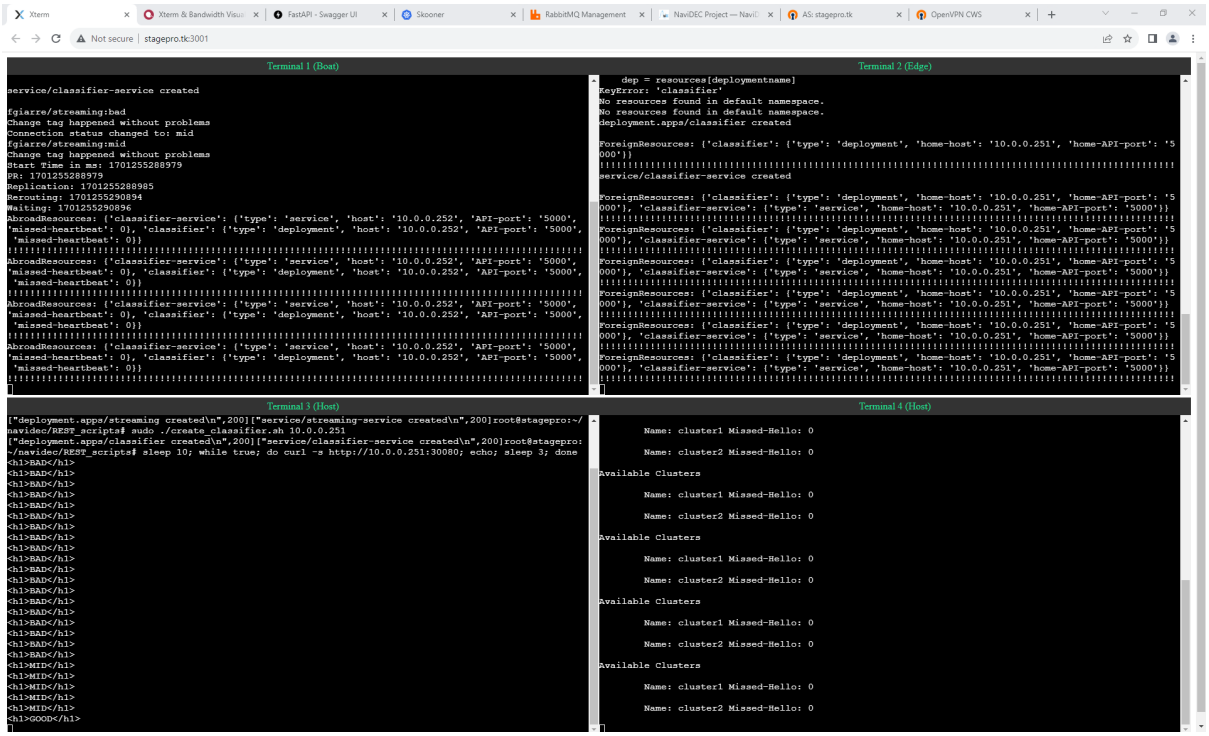

Fig. 3: Display of Four Xterm.js Terminals Embedded in Web Browsers Under High Bandwidth Condition.

2.5.6 In Summary

This web-based terminal subproject exemplifies the IRISA Platform's commitment to user-friendly interfaces, automation, and efficient development practices. Developers and users alike can benefit from the enhanced workflow and interaction capabilities offered by this innovative terminal solution[13].

2.5.7 Code Repository

The source code for this subproject is available in the NaviDEC repository. This provides developers with the opportunity to explore, contribute to, and enhance the web-based terminal interface. One may access the repository at the following path: NaviDEC Repository.

[13] For more information on xterm.js, refer to the official documentation: xterm.js Documentation.

2.6 Bandwidth Visualization

2.6.1 Introduction: Unleashing the Potential of Apache ECharts for Dynamic Bandwidth Visualization

In the dynamic realm of data representation and interactive visualizations, the IRISA Platform is at the forefront of leveraging cutting-edge technologies to craft a seamless and intuitive user experience. A notable stride in this pursuit of excellence manifests in the strategic adoption of Apache ECharts, a robust JavaScript charting library renowned for its versatility and rich feature set.

Apache ECharts is the fundamental tool for visualizing dynamic bandwidth within the platform. It provides users with a clear and detailed view of bandwidth performance. The Stage Speed Gauge, a charting element of Apache ECharts, coupled with real-time data updates, enhances the user's understanding of network bandwidth. This innovative solution ushers in a new phase of network bandwidth management, offering users an unparalleled level of insight and control.

2.6.2 Why Apache ECharts?

Initially, the platform lacked real-time bandwidth visualization and an API to read the bandwidth. Instead, there was a JSON text file that allowed manual bandwidth changes. However, this method did not provide a way to know the actual value unless the file was manually opened and checked. This led to the integration of Apache ECharts, a decision driven by its ability to adapt seamlessly to diverse data sources, its support for interactive features, and its capacity to render visually appealing charts.

The file has been replaced with an API called "Bandwidth Rotator" that generates a preset number every minute. A manual mode has also been integrated as a control bar on the same page. This allows manual control over the auto mode of the Bandwidth Rotator API, serving as an extra feature for platform demonstration in case the end user wants to give it a try. Both auto and manual modes will reflect the change in real time to the bandwidth gauge, an ECharts visualization. The web browser incorporates the API and a socket for backend connection.

Apache ECharts has several advantages over other charting libraries like Highcharts, Google Charts, D3.js, Plotly.js, and amCharts:

1. **License:**

   Apache ECharts is an open-source library under the Apache License 2.0, allowing free usage and modification for both personal and commercial purposes. On the other hand, Highcharts is a commercial library that requires the purchase of a license for commercial use[14], and amCharts also has a proprietary license[Page 33, 14].

2. **Compatibility:**

   ECharts supports a wider range of platforms including Browser, mobile devices, and node.js, making it versatile for various applications[15]. Google Charts primarily focuses on browser compatibility but does not have native support for node.js[16].

3. **Customization Options:**

   ECharts provides a rich set of customizable options, allowing developers to create unique and visually appealing charts. It has a wide range of chart types, themes, and animation effects[15].

4. **Performance and Rendering:**

   ECharts is optimized for large-scale data visualization and provides smooth rendering even with thousands of data points. It also utilizes a lightweight canvas-based rendering engine for better performance[15][17].

5. **Community Support and Documentation:**

   ECharts has a growing and active community, with regular updates and improvements. The library is well-documented, providing extensive guides and examples to help developers get started quickly[17].

[14] Head-to-Head: Echarts vs Highcharts Analysis - Moiva.io. https://moiva.io/?npm=echarts+highcharts.
[15] ECharts vs Google Charts | What are the differences? - StackShare. https://stackshare.io/stackups/echarts-vs-google-charts.
[16] ECharts vs Highcharts | What are the differences? - StackShare. https://stackshare.io/stackups/echarts-vs-highcharts.
[17] Comparing the most popular JavaScript charting libraries. https://blog.logrocket.com/comparing-most-popular-javascript-charting-libraries/.

6. **Ease of Use:**

   ECharts is considered more beginner-friendly compared to D3.js2. Its high-level API and built-in chart types make it easier to create basic visualizations without extensive coding knowledge[18].

2.6.3 Benefits of Apache ECharts Bandwidth Visualization within IRISA Platform

The integration of Apache ECharts for bandwidth visualization within the IRISA platform offers several benefits:

1. **Intuitive Representation:**

   The Stage Speed Gauge effectively translates complex bandwidth metrics into a visually intuitive language. This allows users to quickly interpret the gauge's readings, providing them with immediate insights into the quality of their LTE internet connection. The gauge indicates whether the bandwidth is low, medium, or high. This intuitive representation simplifies the understanding of intricate LTE internet connection metrics, making it easier for users to monitor and manage their connection quality.

2. **Real-Time Bandwidth Monitoring:**

   Leveraging Apache ECharts' real-time capabilities, the Stage Speed Gauge ensures that users receive instantaneous feedback on bandwidth fluctuations.

3. **Visual Appeal:**

   The gauge's visual clarity promotes user engagement by presenting bandwidth data in a format that is both visually appealing and comprehensible.

4. **Adaptability:**

   Apache ECharts seamlessly adapts to diverse data sources, making it a versatile tool for various applications.

5. **Interactive Features:**

   The support for interactive features enhances user engagement and facilitates a more intuitive understanding of data.

6. **Customizable Thresholds:**

   Tailored to the unique needs of the IRISA Platform, the Stage Speed Gauge supports customizable thresholds, enabling users to set parameters that align with specific performance criteria. This customization empowers users to define what constitutes optimal network speed.

7. **Manual Control:**

   The addition of a manual mode allows users to take control over the auto mode of the Bandwidth Rotator API, providing an extra feature for platform demonstration.

8. **Backend Connection:**

   The web browser incorporates the API and a socket for backend connection, ensuring smooth data flow and timely updates.

9. **Responsive Web Design (RWD):**

   The platform is designed with RWD principles, ensuring optimal viewing and interaction experience across a wide range of devices.

10. **Integration with Python and Node.js:**

    The platform integrates seamlessly with Python and Node.js, expanding its capabilities and allowing for more complex operations.

11. **Fully Automated Execution:**

    The platform supports automated execution, reducing manual intervention and increasing efficiency.

These enhancements significantly improve the platform's functionality and user experience, making it a more powerful tool for network monitoring and management.

[18] D3.js vs ECharts | What are the differences? - StackShare. https://stackshare.io/stackups/d3-vs-echarts.

2.6.4 In-Depth Examination of Bandwidth Visualisation of IRISA Platform

Here is an overview of the functionalities provided by the bandwidth visualization subproject:

- **HTML Structure (`bandwidthvisualization/index.html`):**
  - Defines the structure of the HTML page for bandwidth visualization.
  - Includes necessary external libraries such as xterm.js, Socket.IO, noUiSlider, and ECharts.
  - Utilizes inline and external styles for layout and aesthetics.
  - Sets up two main sections: a terminal (*#terminal*) and a chart (*#chart-container*).
  - Integrates a slider (*#slider*) for adjusting bandwidth manually.
  - Includes JavaScript scripts for terminal functionality (*bandwidthvisualizationChart.js* and *sliderBandwidthController.js*).
- **Server Configuration (`bandwidthvisualization/server.js`):**
  - Creates an Express server with Socket.IO for handling WebSocket connections.
  - Listens on port 3002.
  - Serves static files, including the HTML file.
  - Manages WebSocket connections to facilitate real-time communication with clients.
  - Spawns a pseudo-terminal (pty) for command execution and communication with the client terminal.
  - Receives and logs RabbitMQ credentials and a unique token from the server.
  - Provides an endpoint (*/hostname*) to get the server's hostname.
- **Bandwidth Visualization Chart (`bandwidthvisualization/bandwidthVisualizationChart.js`):**
  - Retrieves the server hostname asynchronously.
  - Initializes an ECharts instance for bandwidth visualization.
  - Configures a gauge chart with two series: one for bandwidth and another for outer labels.
  - Establishes a WebSocket connection with the server for real-time updates.
  - Listens for 'bandwidth_update' events and updates the chart accordingly.
  - Handles chart resizing on window resize events.
- **Slider Bandwidth Controller (`bandwidthvisualization/sliderBandwidthController.js`):**
  - Retrieves the server hostname asynchronously.
  - Creates a noUiSlider instance for manual bandwidth adjustment.
  - Defines a range, step, tooltips, and formatting for the slider.
  - Listens for slider changes and updates the server with the new bandwidth value using a POST request.
- **Backend Script (`bandwidthvisualization/kubectl_get_all.py`):**
  - Periodically retrieves the current bandwidth from the server using an API request.
  - Executes Docker commands for fetching status information from different containers (*Boat* and *Edge*).
  - Displays information about orchestrator status, bandwidth, Curl, Boat Cluster, and Edge Cluster status.
  - Utilizes the Rich library for console output formatting.
  - Retries on error and sleeps for 5 seconds between iterations.
- **Miscellaneous:**
  - Uses various external libraries and dependencies, such as xterm.js, ECharts, Socket.IO, noUiSlider, Rich (Python library), and Express.

This project integrates a web-based terminal (*xterm.js*), real-time bandwidth visualization (*ECharts*), and manual bandwidth control through a slider. The server communicates with the client, updating the visualization and receiving input through WebSockets. The backend script periodically fetches information, orchestrates Docker commands, and displays formatted output in the console. This subproject provides a comprehensive solution for monitoring and controlling bandwidth network-related aspects.

2.6.5 Illustration: Bandwidth Visualization Embedded in Web Browser for IRISA Platform

Here are five figures that illustrate bandwidth visualization at various values, both in Auto mode and Manual mode:

The illustrations showcases a web browser with an open tab directed to the server name, followed by its port address, specifically, *http://turing:3002*. The browser window is divided into two main sections:

1. **Left Half - Web Terminal:** Powered by *xterm.js*, this section displays several key components:
    - **Bandwidth Status Section:** Displays the current bandwidth value.
    - **Orchestrator Status:** Lists all active clusters, such as cluster1 and cluster2.
    - **Curl Status:** Shows the streaming tag according to the current bandwidth speed, categorized as BAD, MED, or GOOD bandwidth tag.
    - **Boat Cluster Status:** Displays both streaming and classifier resources when the bandwidth is low or medium, and only streaming when the bandwidth is high.
    - **Edge Cluster Status:** Displays only classifier resources when the bandwidth is high, and indicates no resources when the bandwidth is low or medium.
2. **Right Half - Bandwidth Visualization and Controller:** This section contains:
    - **Bandwidth Gauge Visualization:** Located at the top and powered by *Apache ECharts*, it represents the real-time bandwidth value. The gauge is color-coded into three categories:
        - Red: Indicates a low bandwidth range where the classifier resource will stay in 'Boat'.
        - Blue: Indicates a medium bandwidth range where the classifier resource will also stay in 'Boat'.
        - Turquoise: Indicates a high bandwidth range where the classifier resources will move to 'Edge'.

      In all three color cases, the streaming resources are always located on the 'Boat' cluster1.
    - **Slider Bandwidth Controller:** Located at the bottom and powered by *noUiSlider*, it allows for changing the bandwidth value in Manual Mode for platform demonstration purposes. It operates as follows:
        - A value of 0 is the default and indicates Auto Mode, where the bandwidth visualization gauge will read from the Bandwidth Rotator API.
        - Any changed value will trigger Manual Mode, cancel reading from the Bandwidth Rotator API, and set the value from the slider Bandwidth Controller. This will update the bandwidth visualization gauge. The set value will remain unless it is changed to another value. Depending on whether the changed value is low, medium, or high, the classifier resource will either stay at 'Boat' or move to 'Edge'.
        - Returning to 0 will switch from Manual Mode back to Auto Mode to read the value from the Bandwidth Rotator API.

This structure provides a comprehensive overview of bandwidth visualization and its impact on classifier resources.

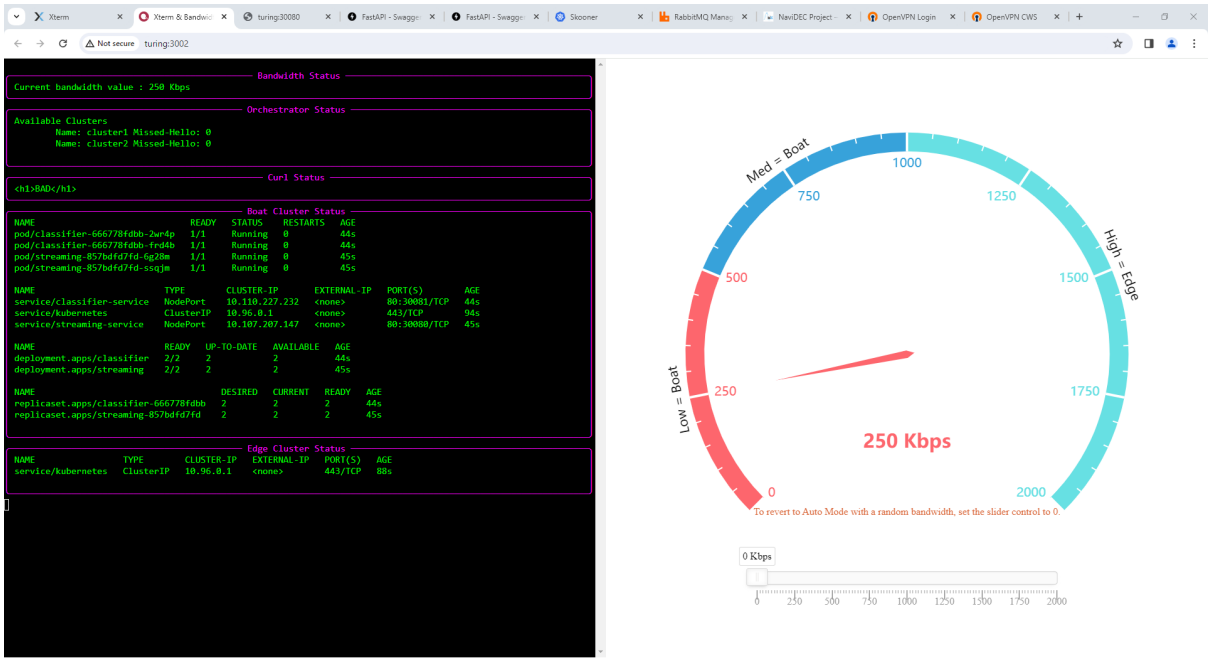

Fig. 4: Display of Web Browser Bandwidth Visualization in Auto Mode Under Low Bandwidth Conditions.

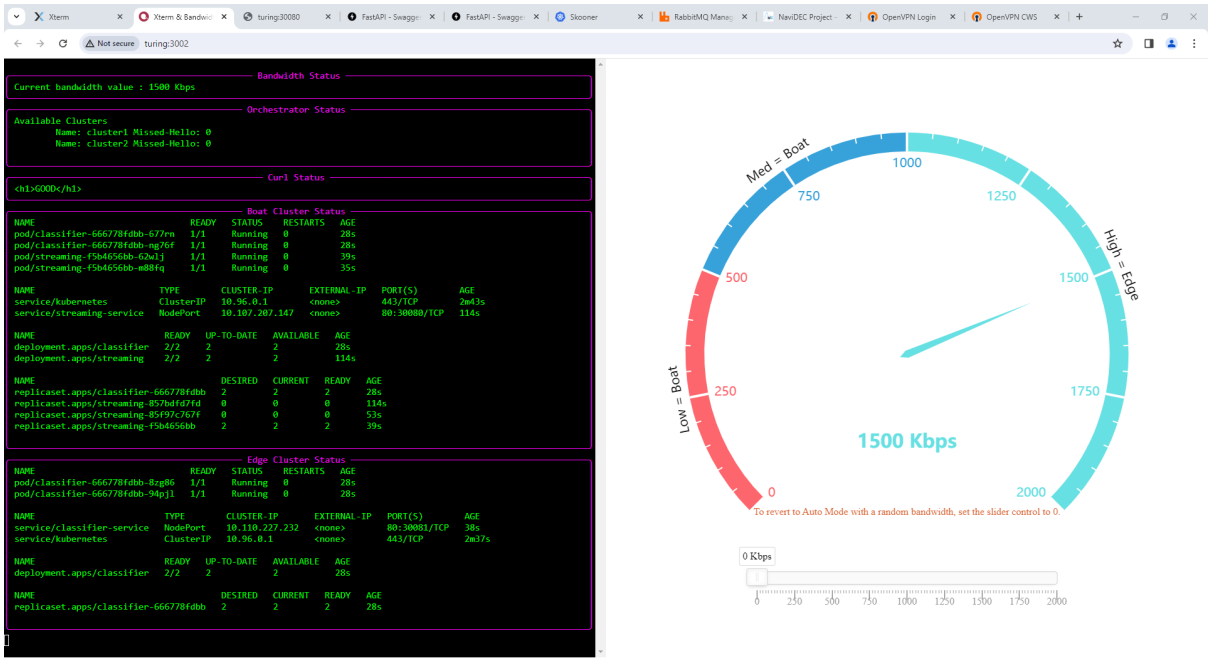

Fig. 5: Display of Web Browser Bandwidth Visualization in Auto Mode Under High Bandwidth Conditions.

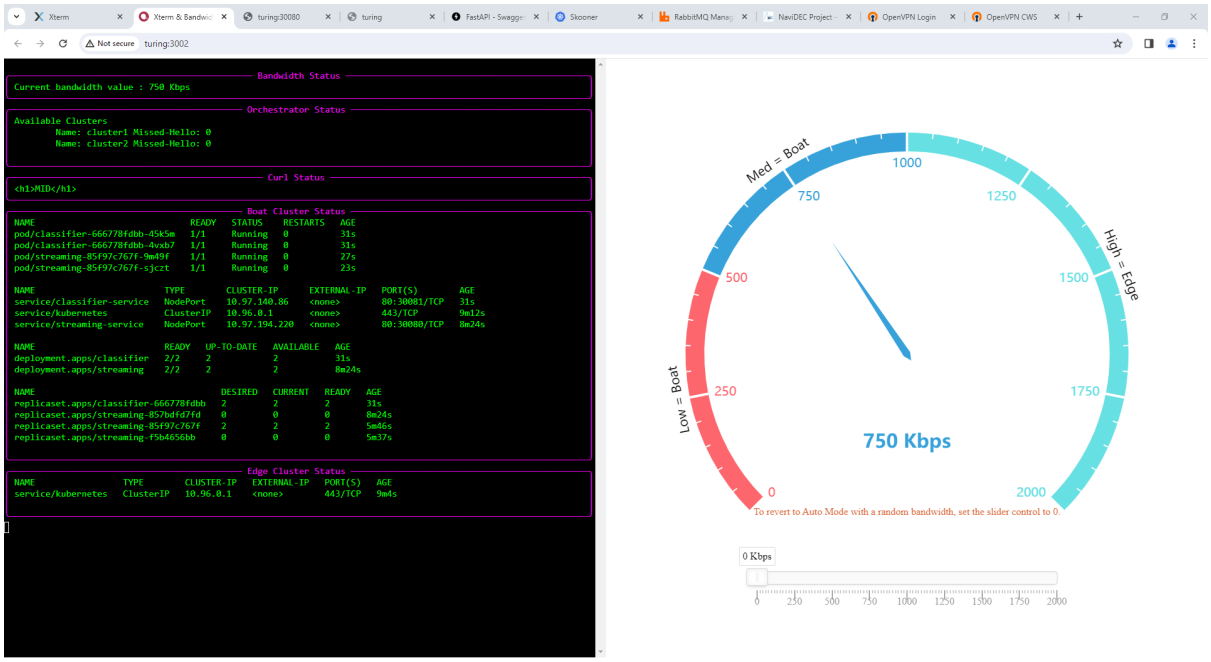

Fig. 6: Display of Web Browser Bandwidth Visualization in Auto Mode Under Medium Bandwidth Conditions.

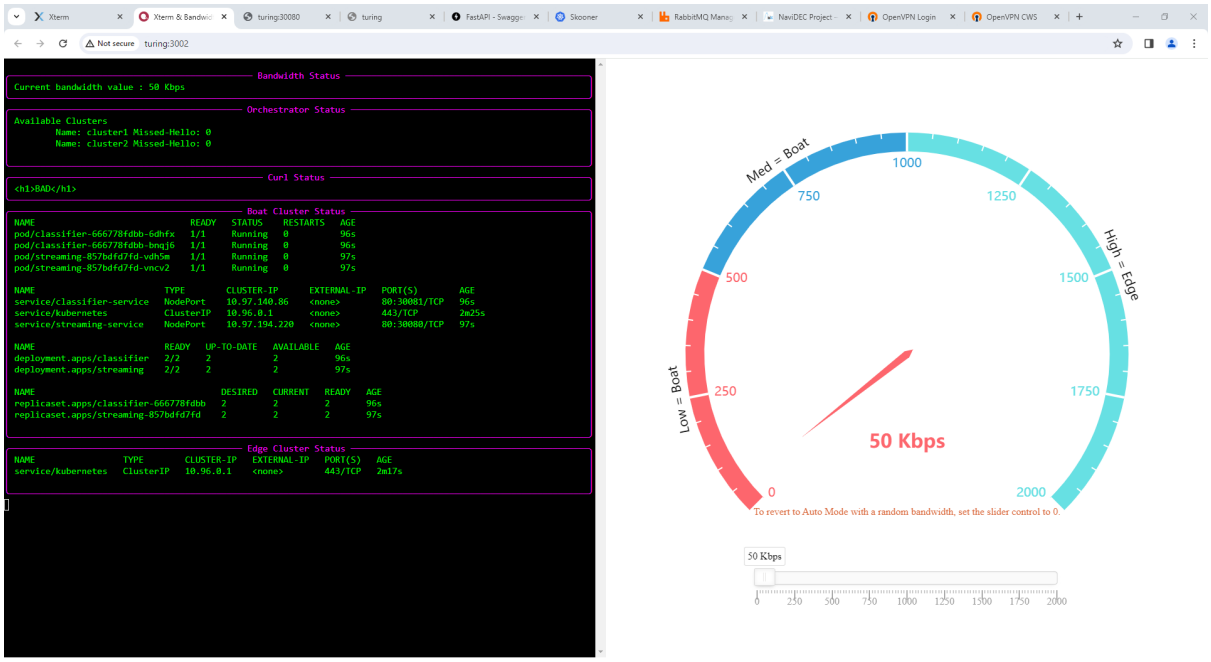

Fig. 7: Display of Web Browser Bandwidth Visualization in Manual Mode Under Low Bandwidth Conditions.

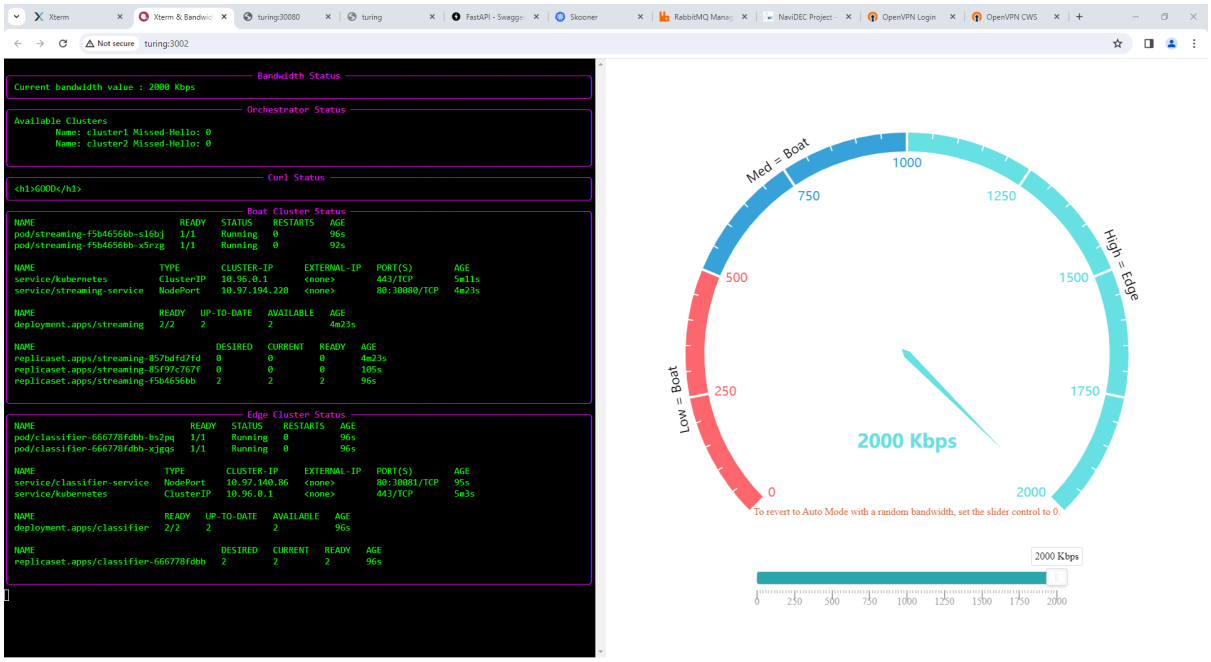

Fig. 8: Display of Web Browser Bandwidth Visualization in Manual Mode Under High Bandwidth Conditions.

2.6.6 In summary

Apache ECharts offers a combination of open-source licensing, wide platform compatibility, extensive customization options, optimized performance for large-scale data visualization, and a growing community. These features make it a strong contender when choosing a charting library for IRISA Platform Project.

2.6.7 Code Repository

The source code for this subproject is available in the NaviDEC repository. This provides developers with the opportunity to explore, contribute to, and enhance the Bandwidth Visualisation basecode. You can access the repository at the following path: NaviDEC Repository.

2.7 Streamlined Integration of Skhooner Kubernetes Dashboard

2.7.1 Introduction: Enhancing Cluster Management with Skhooner

The IRISA Platform extends its capabilities by seamlessly incorporating the Skhooner Kubernetes Dashboard. This integration aims to provide a comprehensive solution for visual resource management within the Kubernetes cluster. The decision to adopt Skhooner aligns with the project's commitment to efficiency, user-friendliness, and streamlined cluster interaction.

2.7.2 Why Skooner?

In the original version of the IRISA Platform, Kubernetes Cluster Management could only be managed through the *kubectl* tool as a command-line interface or by using the NaviDEC REST API. These methods were not considered user-friendly for inexperienced end users. To enhance Cluster Management, Skooner was integrated.

Skooner lies at the core of the IRISA Platform, chosen for its ability to provide comprehensive cluster management, real-time updates, and health visualization. The decision to utilize Skooner aligns with the project's commitment to delivering a user-friendly and efficient Kubernetes management experience.

2.7.3 Benefits of Skooner in the IRISA Platform

Here are some benefits of using Skooner within the IRISA platform:

1. **Full Cluster Management:**

   Skooner enables the management of various components of the cluster, including Namespaces, Nodes, Pods, Replica Sets, Deployments, Storage, and RBAC.

2. **Real-Time Updates:**

   Skooner is always live, meaning there's no need to refresh pages to see the latest cluster status.

3. **Health Visualization:**

   Real-time charts facilitate the quick identification of poorly performing resources and provide a visualization of cluster health at a glance.

4. **Easy CRUD and Scaling:**

   Skooner provides inline API docs to easily understand what each field does.

2.7.4 In-Depth Examination of Skooner Integration in the IRISA Platform

- **Visual Resource Management:**

  Skhooner's intuitive dashboard offers a user-friendly interface for monitoring and controlling various components within the Kubernetes cluster. This visual resource management simplifies the understanding and oversight of cluster activities, enhancing the overall user experience.

- **Seamless Token Retrieval:**

  The dedicated function, *skhooner_get_deployment_token*, plays a crucial role in orchestrating the deployment of Skhooner within the IRISA Platform. This multifaceted function not only deploys Skhooner but also extracts the deployment token, ensuring secure access to the Kubernetes Dashboard within Docker containers. The extracted token is prominently displayed in the SSH terminal, providing users with immediate access. Additionally, the token is saved and accessible through the web browser console, found in the Xterm & Bandwidth Visualization tab of the IRISA Platform. Users can conveniently access this log by pressing F12 to open the browser console and navigating to the specified location. This integration enhances both security and accessibility, offering a streamlined and transparent experience for users interacting with the Kubernetes Dashboard.

- **Multiprocessing for Port Forwarding and Pod Readiness:**

  To optimize performance, a multiprocessing approach has been implemented for port forwarding and pod readiness checks. The *PortForwardProcess* efficiently manages the execution of kubectl port-forward commands for various components, ensuring smooth and dynamic integration.

2.7.5 Illustration: Skooner Dashboard Integration within IRISA Platform

This figure illustrates the Skooner Kubernetes Dashboard. On the left bar of this webpage, we can see the following sections: CLUSTER, WORKLOADS, SERVICES, REPLICAS, PODS, INGRESSES, CONFIG, STORAGE, ACCOUNTS, PROFILE, and APPLY. Each of these sections provides extensive information about the related elements. The 'Apply' section can be used to deploy a deployment or service as a YAML file. In the middle of the webpage, we can see the number of WORKLOADS and PODS. We can also see the names of deployments, such as 'classifier' and 'streaming', along with their related namespaces.

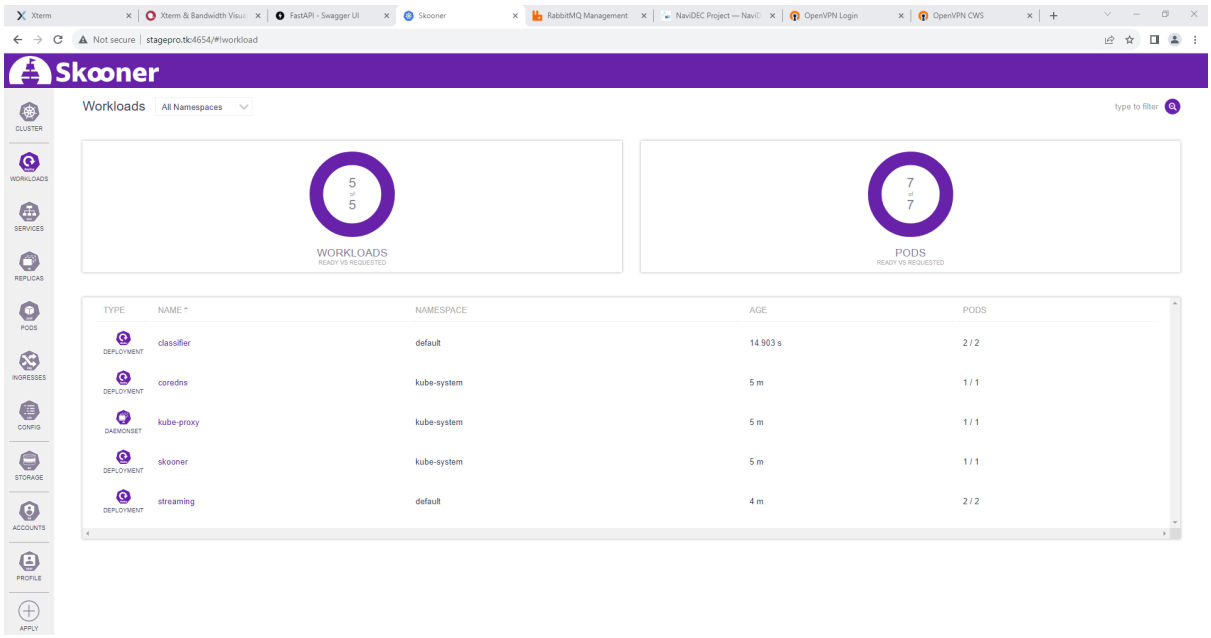

Fig. 9: Overview of the Skooner Kubernetes Dashboard User Interface for Boat, referred to as Cluster1.

2.7.6 In Summary

The integration of the Skhooner Kubernetes Dashboard enhances the IRISA Platform's cluster management capabilities. Developers and users can leverage Skhooner's visual resource management, simplified token retrieval, and multiprocessing functionalities to achieve a more efficient and user-friendly Kubernetes cluster interaction. The project's commitment to innovation and practical solutions is further by the seamless incorporation of Skhooner.

2.7.7 Code Repository

For more details, Click here to view the skooner integration basecode.

2.8 Enhanced Framework Migration: Flask to FastAPI

2.8.1 Introduction: Elevating Performance and Functionality with FastAPI

The IRISA Platform undergoes a transformative framework migration, transitioning from Flask to FastAPI. This strategic move is motivated by a desire to elevate project performance, embrace modern architecture, automate API documentation, and improve error handling.

2.8.2 Why FastAPI?

The decision to adopt FastAPI as the new framework for the IRISA Platform is rooted in its automatic API documentation generation witch not automatically availabe in Flask.

2.8.3 Benefits of FastAPI within IRISA Platform

Here are some benefits of using FastAPI within IRISA platform:

1. **Automated API Documentation:**

   One of the distinguishing features of FastAPI is its capability to automatically produce detailed API documentation. This feature allows developers to interact with the API, even without any pre-existing UI code. It offers developers a clear understanding of the available endpoints and functionalities, thereby simplifying the development workflow.

   To access the auto-generated documentation, navigation to the /docs or /redoc endpoint is required. This presents the Swagger UI, enabling testing of the API endpoints and displaying all the endpoints defined in the application.

   This feature sets FastAPI apart from frameworks like Flask, lacking this automatic documentation generation. While Flask does support documentation, it requires manual setup, which can be seen as a disadvantage.

2. **Optimal Speed:**

   FastAPI is renowned for its exceptional speed, making it one of the fastest Python frameworks. For performance comparisons and more information, please refer to the official FastAPI documentation on performance.

2.8.4 In-Depth Examination of FastAPI migration in the IRISA Platform

Following the migration to the FastAPI framework, there has been an emphasis on enhancing the code for better error handling. Functions have been developed by a dedicated individual to automate the launch process, among numerous other improvements. A comprehensive overview of these enhancements can be found in the following code repository section.

The focus is on the newly developed *BandwidthRotator* API, which replaces the previous method of manually updating bandwidth values via a JSON file. The *BandwidthRotator* class, along with its associated API endpoints, allows for real-time management and updating of bandwidth settings using FastAPI and Socket.IO. Let's delve into the key components:

- **BandwidthRotator Class:**
  - **Initialization:**
    - *self.bandwidth_values*: A list containing predefined bandwidth values.
    - *self.current_bandwidth_index*: Initialized to the last index of *self.bandwidth_values*.
    - *self.shutdown_flag*: An *asyncio.Event* to signal the shutdown of the bandwidth rotation task.
    - *self.manual_mode*: A flag indicating whether the bandwidth rotation is in manual mode.
    - *self.background_task*: An asyncio task that initiates the *rotate_bandwidth* method.

  - **rotate_bandwidth Method:**
    - An asynchronous method that continuously rotates bandwidth values based on a timer.
    - If not in manual mode, it updates the bandwidth index and emits the new bandwidth value to connected clients using Socket.IO.
    - Runs in the background until the *shutdown_flag* is set.
  - **stop_rotating_bandwidth Method:**
    - Stops the bandwidth rotation by setting the *shutdown_flag*.
  - **get_current_bandwidth Method:**
    - Returns the current bandwidth value based on the current index.
  - **set_bandwidth Method:**
    - Accepts a new bandwidth value.
    - If the value is "0," it sets the rotator to automatic mode, immediately rotates to the next bandwidth value, and emits the update.
    - Otherwise, it sets manual mode, updates the bandwidth value, and emits the update.

- **FastAPI Endpoints:**
  - **GET /bandwidth:**
    - Returns the current bandwidth value as a JSON response.
    - Uses the *Depends* function to inject the current bandwidth value from the *BandwidthRotator* class.
  - **POST /bandwidth:**
    - Accepts a new bandwidth value through the request.
    - Updates the bandwidth value in the *BandwidthRotator* class and emits the update.
    - Returns a JSON response indicating successful bandwidth update.
  - **Shutdown Event:**
    - Defines an asynchronous event handler for the FastAPI app shutdown event.
    - Calls the *stop_rotating_bandwidth* method to stop the bandwidth rotation task.
- **Signal Handling:**
  - Defines a signal handler function (*handle_shutdown_signals*) to handle SIGINT and SIGTERM signals.
  - Calls *stop_rotating_bandwidth* on receiving signals to initiate a graceful shutdown.
- **Socket.IO Event:**
  - Defines a Socket.IO event handler for client connection (*connect*).
  - Emits the current bandwidth value to the connected client.
- **Initialization:**
  - Initializes the *BandwidthRotator* class (*bandwidth_rotator*) on FastAPI app startup.
  - Registers the *stop_rotating_bandwidth* method for the app shutdown event.
  - Registers signal handlers for graceful shutdown on SIGINT and SIGTERM.

Overall, this implementation provides a flexible mechanism to dynamically update and rotate bandwidth values in real-time, with support for both automatic and manual modes. Clients connected via Socket.IO receive updates whenever the bandwidth values change. The implementation ensures proper cleanup and shutdown handling.

2.8.5 Illustration: FastAPI automated API docs UI within IRISA Platform

This figure provides an overview of the FastAPI Swagger User Interface for Boat, also known as Cluster1. The interface is designed to handle a variety of requests, including but not limited to GET, POST, and DELETE operations. Each request is associated with a specific endpoint such as 'index', 'nodes', 'resources', 'deployments', 'services', 'pods', and 'bandwidth'. These endpoints serve as gateways for interacting with diverse facets of the system, thereby offering a robust toolset for effective cluster management and monitoring.

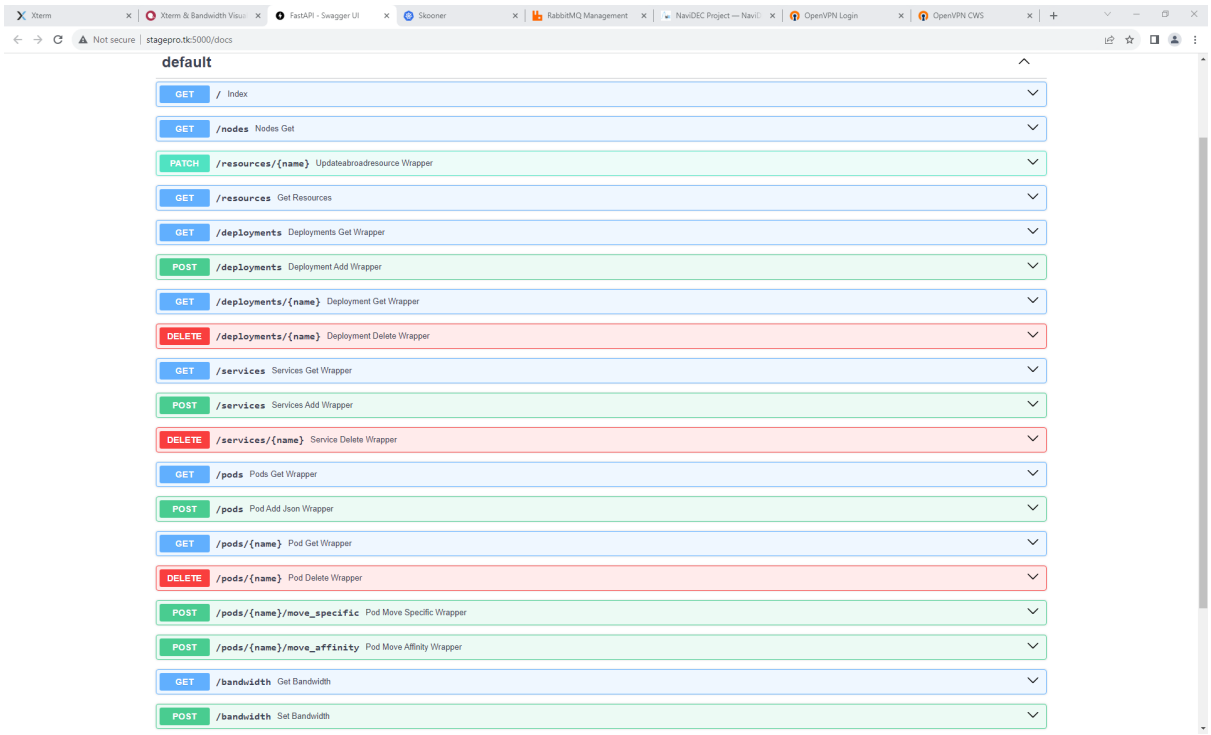

Fig. 10: Display of FastAPI Swagger User Interface for Boat, referred to as Cluster1.

2.8.6 In Summary

The transition from Flask to FastAPI in the IRISA Platform represents a strategic move to achieve optimal performance and automate API documentation. These advancements contribute to a more efficient, reliable, and developer-friendly environment.

2.8.7 Code Repository

For more details, Click here to view the FastAPI framework basecode.

2.9 Comprehensive Logging System

2.9.1 Introduction: Comprehensive Logging System Overview

The *logger.py* script within the IRISA Platform embodies a comprehensive logging system, designed to enhance visibility, traceability, and error monitoring across the platform. This section provides an in-depth overview of the logging system, its rationale, and its pivotal role in the IRISA Platform.

2.9.2 Why Logging Matters ?

In the dynamic and intricate landscape of the IRISA Platform, a robust logging system is indispensable. The *logger.py* script serves as the linchpin for this system, offering a structured approach to capturing critical events, diagnostics, and information flow within the platform. Logging, in this context, goes beyond mere error tracking; it becomes a strategic tool for understanding the platform's behavior, identifying potential improvements, and fostering a proactive approach to system maintenance.

2.9.3 Benefits of Using Logging system within IRISA Platform

Here are some benefits of using Logging system within the IRISA platform:

1. **Visibility and Traceability:**

   The IRISA platform comprises multiple components distributed across clusters. Logging ensures visibility into the activities of each component, allowing developers and administrators to trace the flow of information, understand interactions, and identify potential bottlenecks.

2. **Error Detection and Diagnosis:**

   Comprehensive logging is instrumental in detecting errors and diagnosing issues promptly. The structured logs generated by *logger.py* provide contextual information about the environment, facilitating a quicker and more accurate diagnosis of problems.

3. **Performance Monitoring:**

   Monitoring the platform's performance is a critical aspect of ensuring optimal functionality. The logging system captures timestamps, cluster identifiers, and other relevant data, enabling the monitoring of response times, resource utilization, and overall system health.

4. **Centralized Log Storage:**

   The *logger.py* script facilitates centralized log storage by creating a log file (*navidec_logger.log*). This centralized approach streamlines log analysis, troubleshooting, and monitoring across the entire NaviDEC ecosystem.

5. **Cluster Identification:**

   The logging system incorporates a cluster identification mechanism, associating each log entry with a specific cluster. This feature is vital in a multi-cluster environment, providing context to logs and aiding in the isolation of issues to specific clusters.

6. **Configurable Logging:**

   The logging system is configurable, allowing developers to control the verbosity of logs. By toggling the *logging_enabled* variable, users can seamlessly switch between detailed logging for debugging and minimal logging for production environments.

2.9.4 In-Depth Examination of Logger.py Script within IRISA Platform

The logger.py script encapsulates the logging functionalities within the IRISA Platform. It provides a flexible and configurable logging framework that aligns with the project's goals of visibility, diagnostics, and proactive maintenance.

The script allows users to control the logging level, which can be set to any valid logging level such as INFO, DEBUG, WARNING, ERROR, CRITICAL, etc. This provides flexibility in choosing between detailed logging for development and minimal logging for production.

By default, all logs are written to a file named NaviDEC_logger.log, without being displayed on the console. This approach is designed to keep the console output clean and free from potentially distracting log messages, enhancing the user experience.

Each time the IRISA platform runs, it generates a new log file, ensuring that log data is always up-to-date and relevant.

Here's a breakdown of the key components of the logger.py script:

- **Cluster Identification:**

  The script maps hostnames to cluster identifiers, ensuring that each log entry is tagged with the relevant cluster information.

- **Logger Initialization:**

  The *get_logger* function initializes the logger, setting up the necessary configurations based on the provided name and logging status.

- **Logging Format:**

  A structured logging format is defined, incorporating timestamp, cluster identifier, logger name, log level, and the log message. This format enhances readability and facilitates log analysis.

- **File Handler:**

  If logging is enabled, the script creates a file handler, directing logs to a central log file (*navidec_logger.log*). This file serves as a consolidated repository for logs across different components.

- **Log Level Control:**

  The script allows users to control the logging level, providing flexibility in choosing between detailed logging for development and minimal logging for production.

- **Log Functionality:**

  The script includes a function (*get_current_date*) that generates and logs the current date. This serves as an example of a loggable action within the IRISA Platform.

- **Compatibility with Grafana Loki Logging System:**

  The logs generated by the *logger.py* script can be scraped and visualized using the Grafana Loki logging system. This compatibility enhances the monitoring capabilities of the IRISA Platform, allowing for efficient log aggregation, exploration, and analysis.

In this way, the *logger.py* script becomes an integral part of the IRISA Platform, providing a flexible and configurable logging framework that aligns with the project's goals of visibility, diagnostics, and proactive maintenance.

2.9.5 Illustration: Showcasing the process of log entries being written to the NaviDEC_logger.log file

Here are two figures that illustrate Sample of the NaviDEC_logger.log file, showcasing its content and structure at the beginning and at the end of the file

The sample_navidec_logger.log entries follow a structured format with specific information logged for each entry. Here's an analysis of the structure:

1. **Timestamp:**
   - Format: *YYYY-MM-DD HH:mm:ss,sss*
   - Example: *2023-12-01 19:56:33,240*
2. **Location (Device/Cluster):**
   - Example: *cluster1*
   - Indicates the device or cluster from which the log entry originates.
3. **File Name:**
   - Example: *scripts.config*, *main*, *scripts.utils*
   - Specifies the name of the file containing the code that generated the log entry.
4. **Logger Type:**
   - Example: *INFO*
   - Specifies the type of logger event, such as INFO.
5. **Log Message:**
   - Example: *channel: Setting up channel...*, *Initializing FastAPI: APIs for cluster cluster1 listening on 10.0.0.251:5000. Cluster positioned on the deep-edge*
   - Contains the actual log message describing the activity or event.

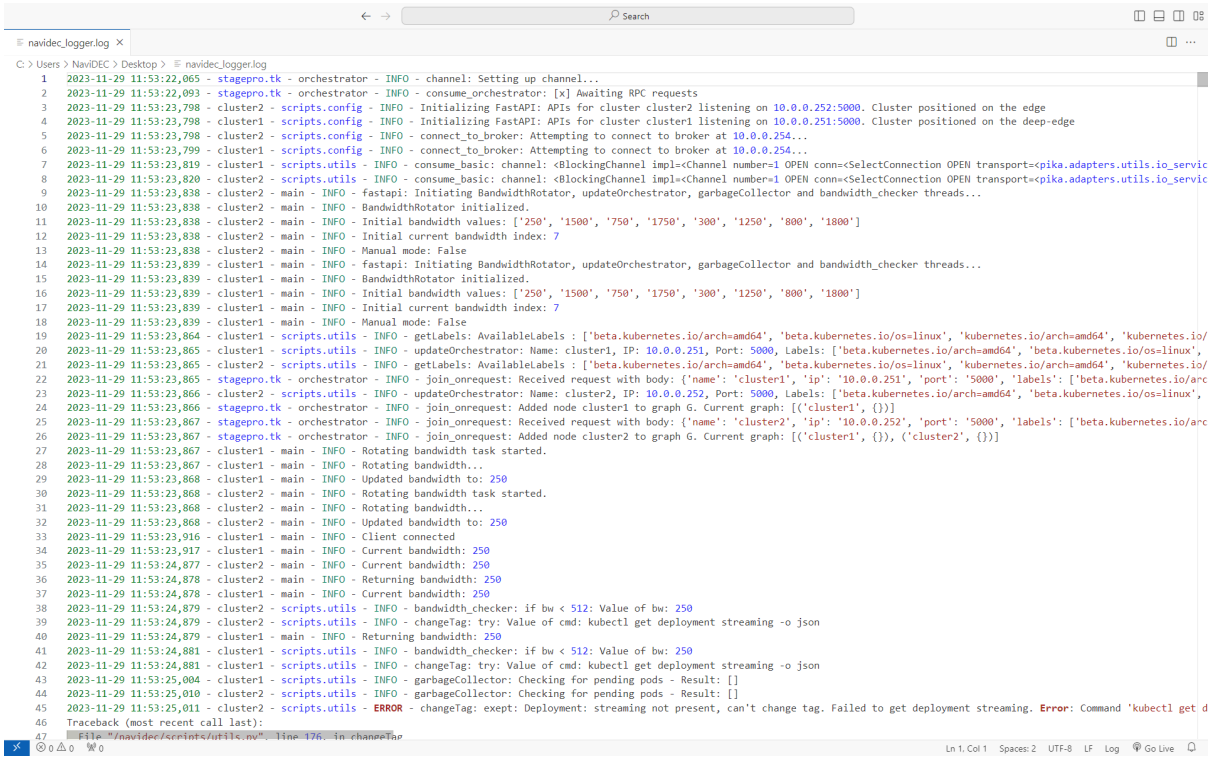

Fig. 11: Sample of the NaviDEC_logger.log file, showcasing its content and structure at the beginning of the file.

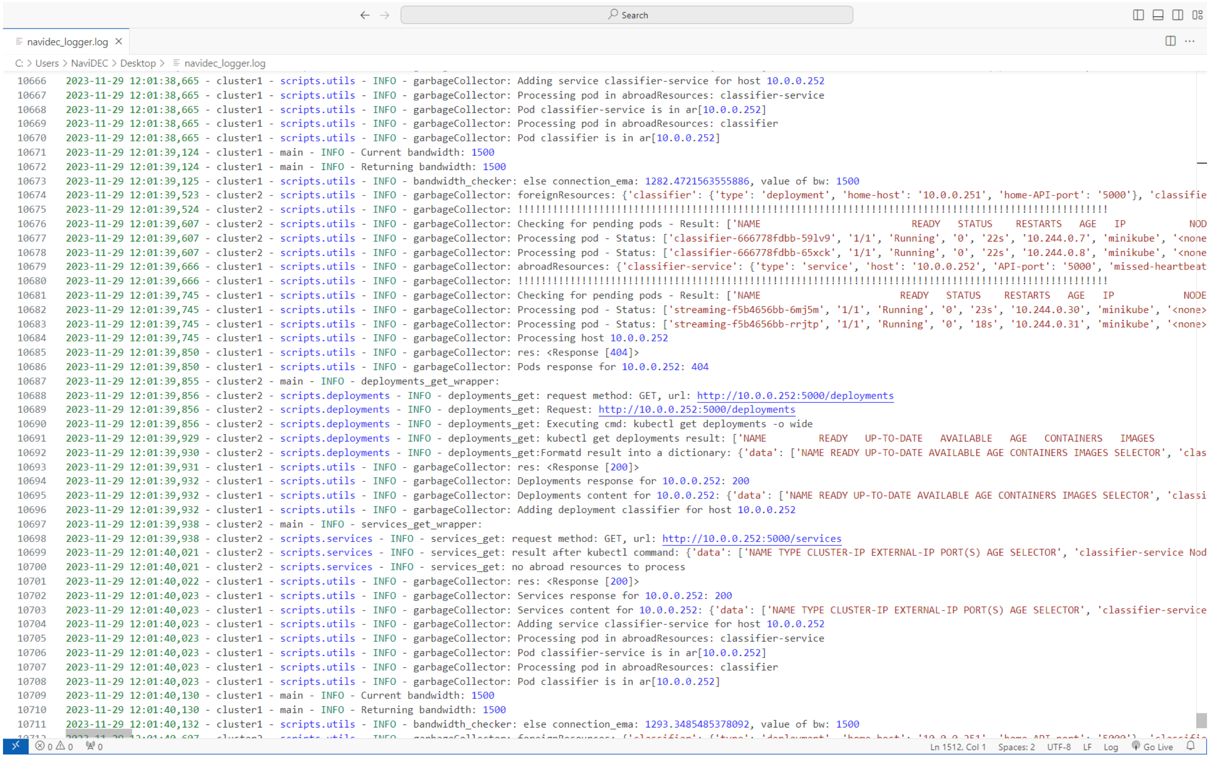

Fig. 12: Sample of the NaviDEC_logger.log file, showcasing its content and structure at the end of the file.

2.9.6 In summary

Each log line in the file has the following structure:

- *Timestamp - Location - File Name - Logger Type - Log Message*

2.9.7 Code Repository

For more details, Click here to view the logger.py source code.

2.10 Diagram Illustration, Enhancing Code Comprehension and Logical Visualization

2.10.1 Introduction: Visualizing the Logic and Flow of the IRISA platform

In the intricate world of software development, the IRISA Platform places a strong emphasis on enhancing code comprehension and logical visualization. This section explores the utilization of diagrams to illustrate the underlying logic and flow of the IRISA platform's architecture. By adopting visual aids, developers can gain a deeper understanding of the system, facilitating collaboration, troubleshooting, and seamless contribution to the codebase.

2.10.2 Why Diagrams ?

The incorporation of diagrams into the IRISA Platform serves multiple purposes, offering developers a visual representation of the platform's architecture, relationships between components, and the flow of data and processes. The advantages include:

1. **Code Comprehension:**

   Diagrams provide a high-level overview of the codebase, making it easier for developers to grasp complex relationships and interactions between different modules and components.

2. **Logic Visualization:**

   Flowcharts and architectural diagrams visualize the logical pathways and data flow within the IRISA platform, aiding developers in understanding the system's behavior.

3. **Collaborative Development:**

   Visual representations act as a common ground for developers, enabling effective collaboration by providing a shared understanding of the codebase and architecture.

4. **Onboarding and Training:**

   New developers joining the project can quickly familiarize themselves with the codebase and architecture through visual diagrams, expediting the onboarding process.

5. **Troubleshooting:**

   When debugging or troubleshooting, developers can refer to diagrams to identify potential points of failure, bottlenecks, or areas requiring optimization.

2.10.3 Benefits of Diagrams for IRISA Platform

Here are some benefits of using Diagrams for IRISA platform:

1. **Enhanced Code Comprehension:**

   Visualizing code through diagrams enhances overall comprehension, allowing developers to see the bigger picture and understand the interplay of different components.

2. **Efficient Communication:**

   Diagrams provide a concise and efficient means of communication, reducing the complexity of conveying intricate architectural details during discussions or code reviews.

3. **Improved Decision-Making:**

   Visual aids aid in decision-making processes by presenting a clear representation of architectural choices, dependencies, and their implications.

4. **Documentation Enhancement:**

   Incorporating diagrams into documentation enriches it, making it more accessible and engaging for developers seeking insights into the platform's architecture.

2.10.4 Types of Diagrams for IRISA Platform

1. **NaviDEC project Overview:**
    - Provides a high-level view of NaviDEC project, focusing on its objectives, stakeholders, and key deliverables, without delving into technical specifics.

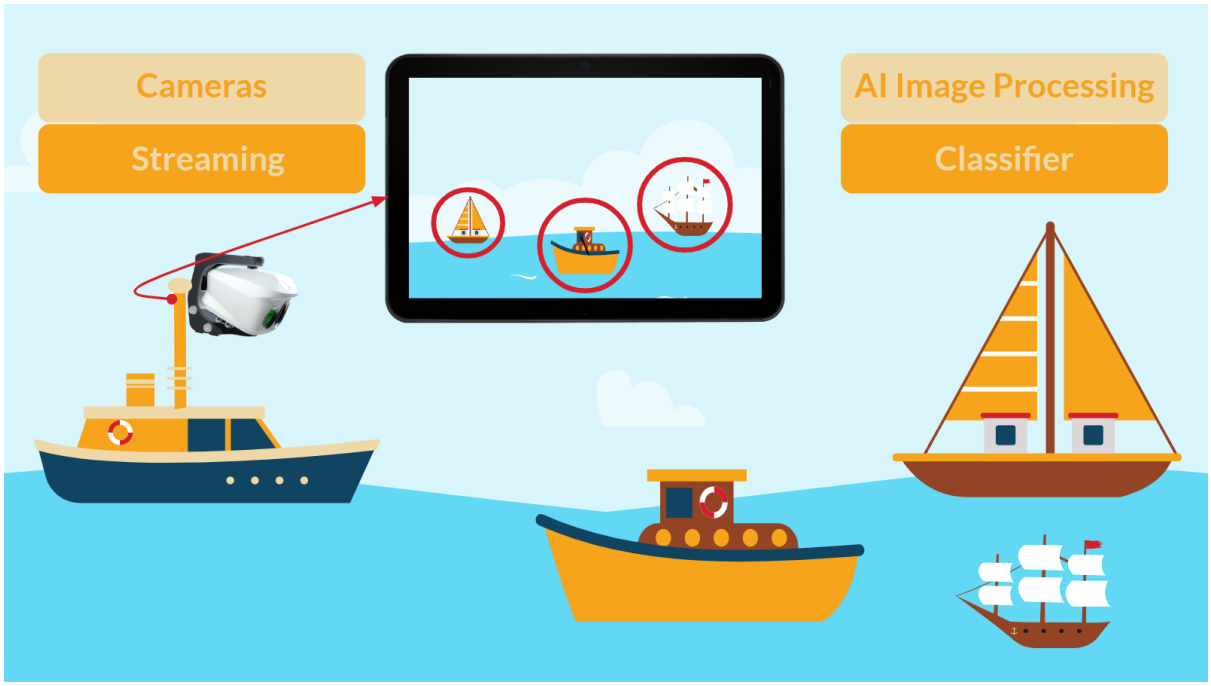

Fig. 13: Display of NaviDEC Work Context.

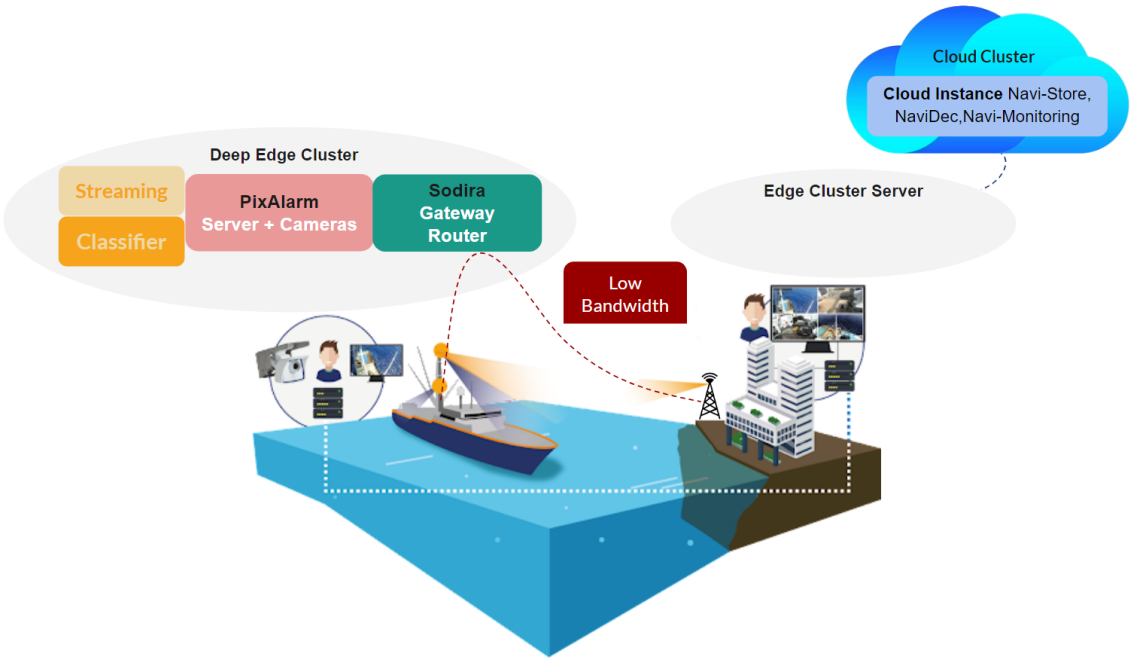

Fig. 14: The NaviDEC use-case is currently operating at a low LTE bandwidth status.

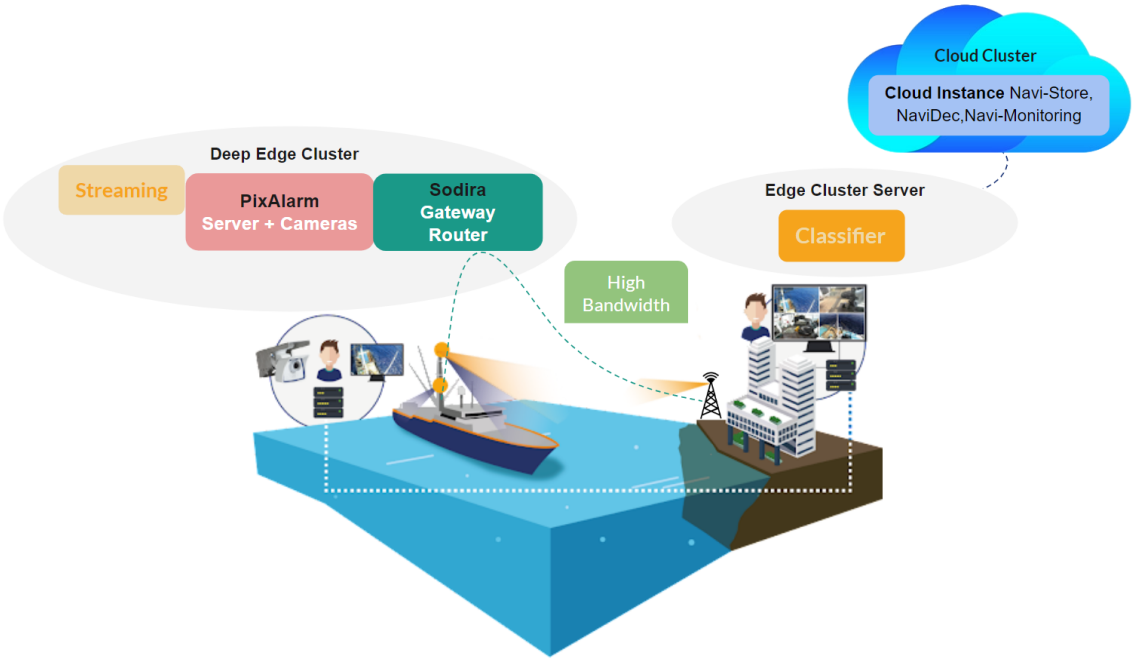

Fig. 15: The NaviDEC use-case is currently operating at a high LTE bandwidth status.

2. **Platform Architecture Diagram:**

   - Illustrates the overall structure of the IRISA platform, including the relationships and dependencies between its various components.

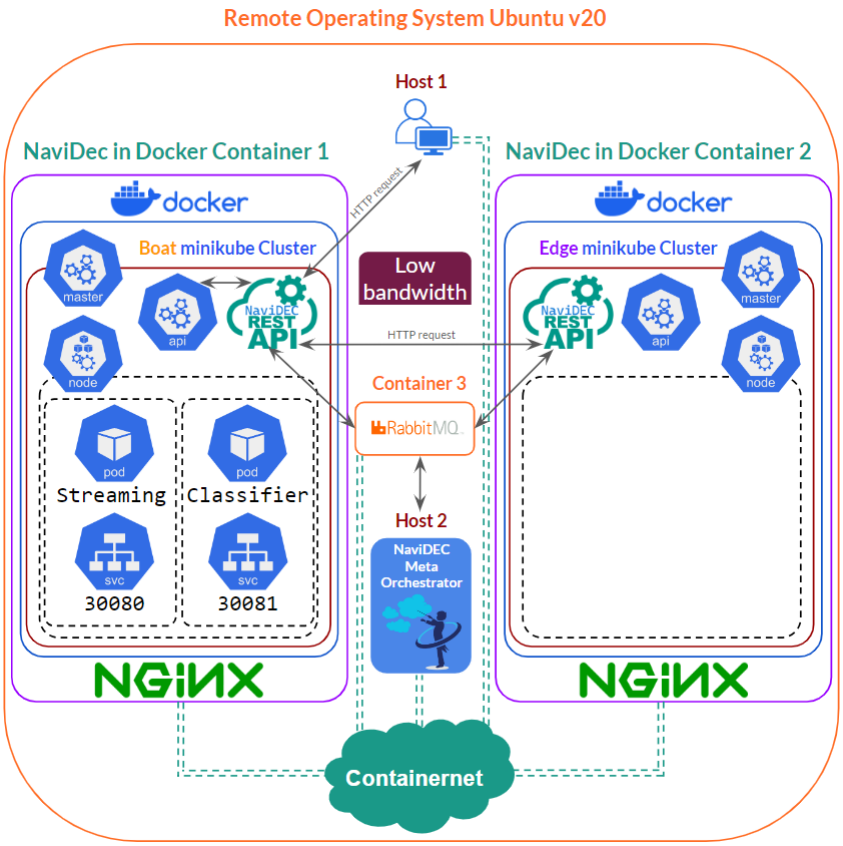

Fig. 16: The IRISA platform is currently operating at a low bandwidth speed.

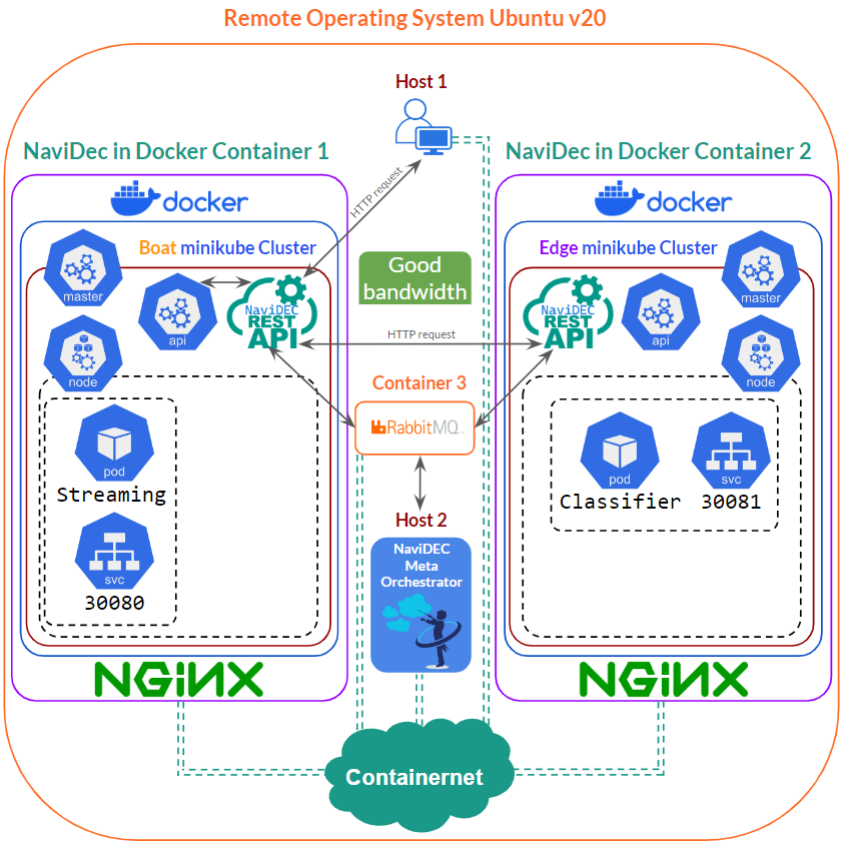

Fig. 17: The IRISA platform is currently operating at a good bandwidth speed.

3. **Flowcharts for Processes:**

   - Visual representations of the flow of processes within the NaviDEC codebase, highlighting decision points and the overall sequence of operations. These diagrams were generated using the Eraser online platform, which can be found at eraser.io.

A high-quality version of the diagram is also available for download: IRISA platform Diagram.svg

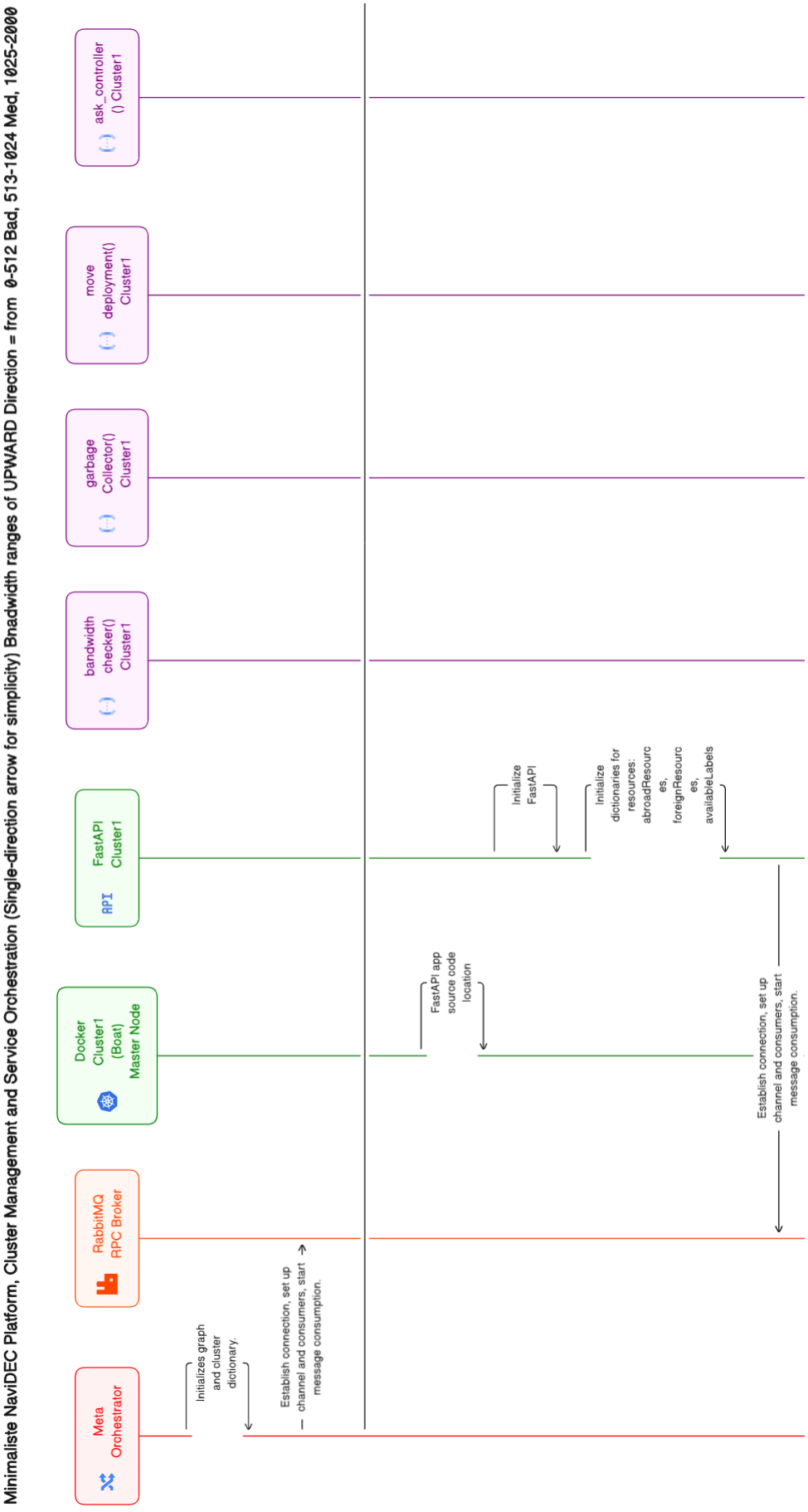

Fig. 18: Segment of the Flowchart Diagram.

2.10.5 In-Depth Examination of Using Eraser Flowchart for IRISA Platform

The IRISA Platform's development workflow diagrams are efficiently managed via EraserDiagramSequence.txt, containing code used to generate flowchart diagrams on eraser.io.

Workflow Steps:

- **Initial Diagram Creation:**

  Diagrams are crafted in Eraser's online platform using code from EraserDiagramSequence.txt.

- **Regular Updates and Revisions:**

  EraserDiagramSequence.txt updates reflect architectural changes as NaviDEC codebase evolves.

- **Integration with Documentation:**

  Eraser seamlessly integrates diagrams into project documentation in SVG and PNG formats.

Eraser Diagram Sequence File:

The EraserDiagramSequence.txt file contains the necessary code to generate diagrams on the Eraser platform. Developers can leverage this file to visualize the workflow, logic, and interactions within the IRISA Platform.

Advantages of Eraser Diagrams in the Workflow:

- **Efficiency and Agility:**

  Eraser enables swift manipulation of diagrams, helping developers adapt to codebase changes.

- **Real-Time Collaboration:**

  Eraser's online platform fosters real-time collaboration, refining NaviDEC workflow jointly.

- **Multiple Output Formats:**

  Eraser exports diagrams to SVG and PNG formats.

- **User-Friendly Interface:**

  Eraser's platform simplifies diagram creation, editing, and updates for developers.

- **Integration with Version Control:**

  Include EraserDiagramSequence.txt in version control to track diagram code changes over time.

2.10.6 Code Repository

For more details, Click here to view the EraserDiagramSequence.txt source code.

2.10.7 In Summary

Diagrams enhance NaviDEC comprehension and foster collaboration. Visual aids facilitate understanding, troubleshooting, and efficient contribution. They align with IRISA Platform's transparency and robust development workflow.

2.11 Sphinx, A Comprehensive Documentation Tool

2.11.1 Introduction: Embracing Sphinx for Detailed Documentation

In the intricate world of software development, the IRISA platform employs Sphinx, a potent open-source tool for generating documentation. Sphinx's wide-ranging capabilities and straightforward usage make it the go-to choice for crafting detailed and aesthetically pleasing documentation. This significantly enhances the IRISA Platform's efficiency and comprehensibility.

2.11.2 Why Sphinx?

Sphinx is the backbone of the IRISA platform's documentation. Renowned for its support for a variety of output formats and extensive cross-references, Sphinx offers semantic markup and automatic links for a multitude of information types, including functions, classes, citations, and glossary terms. Opting for Sphinx is a testament to the project's dedication to providing a user-centric and efficient web-based documentation experience.

2.11.3 Benefits of Using Sphinx for documentationthe for IRISA Platform

1. **Structured Documentation for IRISA Platform:**

   Leveraging Sphinx, Platform crafts organized documentation, enhancing user navigation. Sphinx's hierarchy creates defined sections, subsections, and chapters, contributing to clarity in Platform documentation.

2. **Tailored Output Formats for Enhanced Accessibility:**

   Sphinx, a cornerstone of Platform's documentation strategy, supports multiple output formats including HTML, PDF, ensuring seamless access to documentation. By incorporating Sphinx, Platform enhances accessibility and usability, catering to user preferences.

3. **Cross-Referencing and Linking for Seamless Navigation:**

   Sphinx enables Platform's documentation with cross-referencing between sections, creating hyperlinks and references. This improves coherence, allowing users to navigate between related topics, ensuring a comprehensive understanding of the IRISA Platform.

4. **Efficient Search Functionality:**

   Platform benefits from Sphinx's built-in search functionality, enhancing the user-friendliness and efficiency of the documentation. The search bar enables swift location of relevant information, streamlining exploration of Platform's project details.

5. **Seamless Code Integration for Developers:**

   Sphinx integrates with programming languages, aligning with Platform's code-centric nature, enabling inclusion of code snippets, syntax highlighting, and API documentation generation for understanding technical aspects.

6. **Versioning Support for Ongoing Updates:**

   Sphinx's versioning support is invaluable for Platform during updates and releases, enabling documentation creation specific to IRISA Platform versions. Users access documentation aligning with their working version, ensuring accuracy.

7. **Customization and Theming for Brand Consistency:**

   Platform uses Sphinx's customization options to align documentation with project branding. Sphinx enables a consistent and branded look across documentation, enhancing project identity.

8. **Extensibility with Plugins to Meet Unique Requirements:**

   Sphinx's extensibility through plugins empowers Platform to enhance its documentation with custom features. This flexibility allows Platform to integrate additional tools, unique integrations, or customizations that meet specific project requirements, ensuring documentation evolves alongside the project.

2.11.4 In-Depth Examination of Sphinx Documentation for the IRISA Platform

In the IRISA Platform, Sphinx serves as the cornerstone for creating and exporting documentation in multiple formats, enhancing accessibility and user convenience. The documentation is effortlessly hosted online through GitLab Pages, ensuring continuous deployment through an automated pipeline. Here's a concise summary:

- **HTML Documentation:**

  Utilizing Sphinx, the IRISA Platform exports documentation in HTML format. The hosted HTML content is accessible online through the following link: IRISA Platform, NaviDEC Project Online Documentation

- **PDF Documentation:**

  Sphinx enables the generation of PDF documentation, providing users with a downloadable and printable version. The PDF version is conveniently hosted alongside the HTML content for user convenience. PDF version of this project is available for download: IRISA Platform, NaviDEC Project.pdf

- **Automated Deployment Pipeline:**

  The documentation deployment process is seamlessly integrated into the GitLab pipeline. As soon as content updates are made, the automated pipeline triggers a deployment to GitLab Pages, ensuring that the online documentation is always up-to-date without causing interruptions to the website.

- **Version Synchronization:**

  Both HTML and PDF versions are meticulously built and updated to reflect the most recent content changes. This synchronization guarantees that users have access to the latest version of the documentation, fostering an environment of continuous improvement.

- **Unified Hosting Location:**

  Both HTML and PDF versions of the documentation are hosted in a unified location, providing users with a centralized repository for accessing and downloading the documentation. This centralized approach enhances user convenience and ensures a consistent experience.

- **GitLab Pages Link:**

  For easy access to the online documentation, users can visit the following link: IRISA Platform, NaviDEC Project Online Documentation . This link serves as the primary entry point for exploring the comprehensive documentation provided by Sphinx for the IRISA Platform.

2.11.5 Illustration: Sphinx Documentation Homepage for IRISA Platform

This figure illustrates the Sphinx Documentation homepage for the IRISA Platform hosted on GitLab Pages. The URL located at the top of the page serves as the gateway to the Sphinx Documentation webpage for the IRISA Platform. The webpage is thoughtfully structured, with the main titles of the documentation prominently displayed in the left column. Correspondingly, a comprehensive list of contents related to these main titles can be found on the right. For ease of navigation, one can delve deeper into the sub-titles by simply clicking on any chosen link.

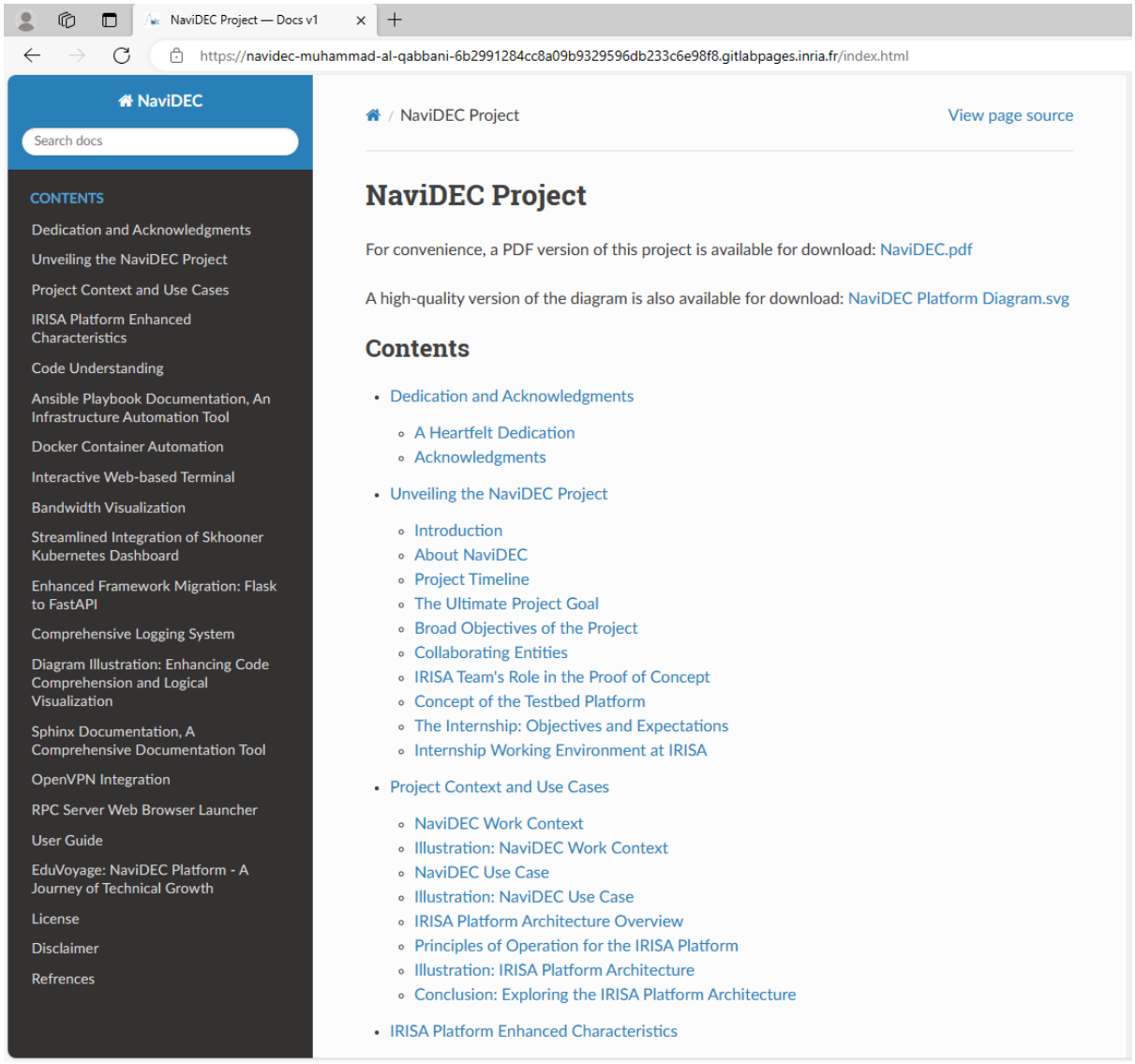

Fig. 19: Sphinx Documentation Homepage for IRISA Platform, hosted on GitLab Pages.

2.11.6 In Summary

Leveraging Sphinx and GitLab Pages, the IRISA Platform seamlessly integrates well-structured documentation into its development pipeline, ensuring heightened transparency and user satisfaction. Sphinx tailors documentation for users and developers, proving invaluable across projects of diverse scales and complexities.

2.11.7 Code Repository

The source code for the Sphinx documentation subproject is available in NaviDEC Repository.

2.12 Automated build and deployment pipeline in GitLab

2.12.1 Introduction: Significance of GitLab Pipeline for Documentation

Documentation is an integral part of any software project, providing insights, guidelines, and instructions for developers and users alike. In the context of Sphinx documentation, the utilization of GitLab Pipeline introduces several advantages, addressing various challenges inherent in the traditional documentation process.

2.12.2 Why Gitlab Pipeline?

1. **Learning Curve and Setup:**

   The learning curve for Sphinx documentation can be steep, with extensive required libraries, extensions, and themes that are approximately 3 gigabytes in size. The setup process can be time-consuming and varies based on the operating system, be it Windows or Linux.

2. **Build and Compile Process:**

   Once all the required files and libraries are downloaded, a build and compile process is necessary whenever there are updates to the documentation. If we need to export HTML and PDF, we need to manually build and compile them separately, which can be time-consuming depending on the power of the CPU used.

3. **Manual Browsing:**

   After the build and compile process, we need to manually browse the HTML files and PDF to ensure the changed files have been updated.

4. **Working from Multiple Locations:**

   Working from multiple locations like the office and home, each with a different operating system, increases the complexity of maintaining consistency in configuration setup.

5. **Cost-Free Deployment:**

   We desire a cost-free method to deploy the work on a website for easy access by relevant individuals.

6. **GitLab Pipeline Solution:**

   The pipeline in GitLab can solve all above issues, and we can get the final result in less than 60 seconds.

2.12.3 Benefits of Using Gitlab Pipeline within IRISA Platform

Here are some benefits of using Gitlab Pipeline within the IRISA platform:

1. **Automation:**

   GitLab CI/CD (Continuous Integration/Continuous Deployment) allows for the automation of the build and deployment process. This means that once you push your changes to the repository, the pipeline automatically builds your project, generates the necessary files, and deploys them.

2. **Consistency:**

   By using a pipeline, you ensure that the build and deployment process is consistent and repeatable. This reduces the risk of errors that can occur when these processes are done manually.

3. **Efficiency:**

   The pipeline can be configured to only build and deploy when certain conditions are met, such as a push to a specific branch like master in our project case. This saves resources by not building and deploying when it's not necessary.

4. **Integration with GitLab Pages:**

   GitLab Pages is designed to work seamlessly with GitLab CI/CD. This means you can easily publish your Sphinx documentation to a web page hosted by GitLab Pages.

5. **Flexibility:**

   GitLab CI/CD is flexible and can be configured to use different tools and environments. This means you can use it to build Sphinx documentation, generate HTML and PDF files, and deploy these files using Docker.

6. **Version Control:**

   Since the pipeline configuration is stored in a *.gitlab-ci.yml* file in your repository, it benefits from version control. This means you can track changes to your pipeline configuration, revert to a previous configuration if necessary, and see who made changes.

The sequence of scripts that GitLab CI/CD runs to accomplish this task is created from a file named *.gitlab-ci.yml*, which you can create and modify. A specific job called *pages* in the configuration file makes GitLab aware that you're deploying a GitLab Pages website.

2.12.4 In-Depth Examination of Gitlab Pipeline within IRISA Platform

Here is an Analysis of *.gitlab-ci.yml* file Configuration:

- **Base Image Configuration:**
  - Utilizes a customized image based on Alpine 3.18 (moimudi/sphinx) as base image of Sphinx.
- **Pages Deployment Stage:**
  - Specifies the name of the stage in the GitLab CI/CD pipeline as 'deploy'.
- **Script Execution Steps:**
  - Navigates to the 'docs' directory.
  - Builds the HTML documentation using Sphinx (*sphinx-build -b html . public*).
  - Builds the PDF documentation using Sphinx and latexmk (*sphinx-build -b latex . _build/latex*).
  - Executes *make -C _build/latex all-pdf* to compile all PDFs.
  - Copies the generated PDF files from *_build/latex/* to the 'public' directory.
- **Artifacts Configuration:**
  - Specifies the paths of the build artifacts to be stored: 'docs/public'.
  - Sets the expiration of artifacts to 'never', meaning they will not expire.
- **Publish Configuration:**
  - Specifies the directory to publish as 'docs/public'.
- **Tags Configuration:**
  - Specifies the use of a runner tagged with 'large' for this job.
- **Rules Configuration:**
  - Defines conditions for job execution:
    - The job runs only when files in 'docs/public/' directory are changed on the master branch.
    - Condition: *$CI_COMMIT_REF_NAME == 'master'*.

Implementation Process:

- The Docker image has been customized to include all required files and has been pushed to Docker Hub.
- The assigned project runners in GitLab have been configured for a private server to handle downloading the Docker image, building and compiling HTML and PDF, and pushing directly to the GitLab page.
- *.gitlab-ci.yml* file sets required information for triggering the pipeline when a change happens to Sphinx file.

2.12.5 Illustration: Showcasing Pipeline Stages Status

This figure illustrates image showcasing the status of Pipeline Stages for building and deploying Sphinx documentation in HTML and PDF formats on the website, executed without any service interruption. We can observe that the pipeline has successfully passed, indicated by the green color. The elapsed time is under 60 seconds, and this operation was performed under the master branch.

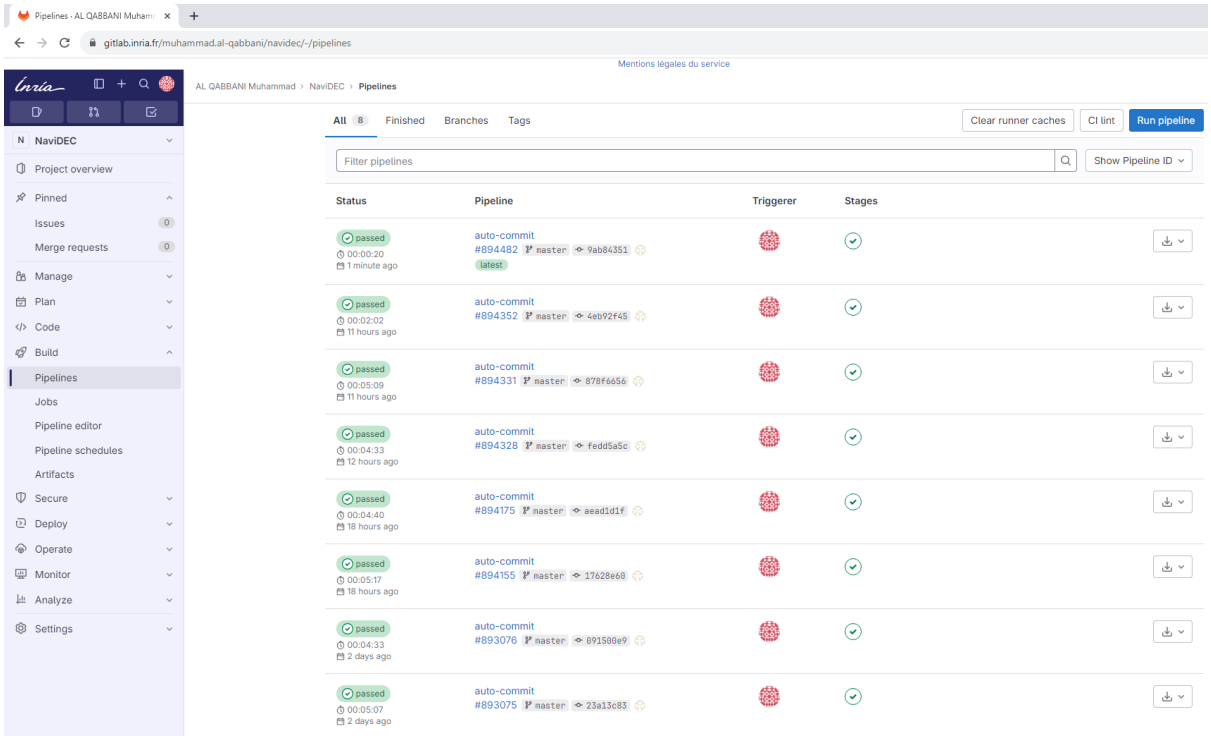

Fig. 20: Display of Pipeline Stages Status for building and deploying Sphinx documentation.

2.12.6 In Summary

GitLab Pages hosts the Sphinx documentation website directly from the NaviDEC GitLab repository, while GitLab CI/CD pipelines automate building, testing, and deploying the code, streamlining development workflows for Sphinx documentation.

2.12.7 Code Repository

For more details, Click here to view the .gitlab-ci.yml source code.

2.13 OpenVPN Integration

2.13.1 Introduction: The Crucial Role of OpenVPN Integration

OpenVPN Integration within the IRISA platform emerges as a crucial augmentation, establishing a secure and dependable communication conduit. This integration plays a pivotal role in enabling the IRISA platform's remote server to establish secure connections with the user's personal computer or laptop, essential for initiating seamless communication. The implementation ensures the integrity and confidentiality of data exchanged between these endpoints. In the ensuing sections, we delve into the rationale behind incorporating OpenVPN, shedding light on its pivotal role in facilitating the IRISA platform's capability to ping and communicate with the user's personal computer or laptop. Additionally, we explore how OpenVPN contributes to streamlining automation, enabling the effortless launch of web browsers on the user's device from IRISA platform.

2.13.2 Why OpenVPN?

To utilize the IRISA platform, one must establish a connection to a remote server running Ubuntu 20.xx. This can be achieved from a personal computer or laptop, irrespective of whether it operates on a Windows or Linux system, by initiating an SSH terminal to connect to the remote IRISA platform.

Typically, the personal computer or laptop we use to access the remote server of the IRISA platform is connected to the internet via Network Address Translation (NAT) with a private IP such as 192.168.1.1. This IP, being non-routable through the internet, prevents the possibility of direct pinging from the IRISA platform's remote server to a private IP address, such as 192.168.1.1, assigned to the personal computer or laptop. This is a result of the inherent limitations of internet protocols.

To overcome the limitations of internet protocols, the OpenVPN program is used to establish a direct connection between the remote server of the IRISA platform and the personal computer or laptop. In this setup, the personal computer or laptop is assigned a private VPN IP, specifically 10.10.10.10, enabling effective communication with the IRISA platform. This configuration notably enhances the ability to ping from the IRISA platform's remote server to the personal computer or laptop used for access.

The main objective of this setup is to automate the process of opening a web browser on the personal computer or laptop that is used to access the IRISA platform.

This setup enables the remote server of the IRISA platform to send a command when it's ready. This command initiates a local web browser, such as Google Chrome for Windows or Firefox for Linux, on the specified personal computer or laptop. The web browser will automatically open all necessary tabs for the URLs used by the IRISA platform. This includes, but is not limited to, the xterm.js interactive terminal and bandwidth visualization.

2.13.3 Benefits of OpenVPN within IRISA Platform

Here are some benefits of using OpenVPN within IRISA platform:

1. **Secure Connection:**

   OpenVPN provides a secure and encrypted connection over the internet between two endpoints.

2. **Overcoming Limitations:**

   Enabling direct pinging from the IRISA platform's remote server to the PC or laptop.

3. **Private Communication:**

   The PC or laptop is assigned a private VPN IP, enhancing privacy and security of communication.

4. **Ease of Access:**

   It facilitates easy access to the IRISA platform from any location, given the availability of internet connection.

5. **Versatility:**

   OpenVPN supports a wide range of operating systems, making it versatile for different users and systems.

6. **Automation:**

   It automates opening a local web browser with necessary tabs for URLs used by the IRISA platform.

2.13.4 In-Depth Examination of OpenVPN Integration within IRISA Platform

The integration of OpenVPN within the IRISA platform involves a series of carefully orchestrated steps. The following is a detailed examination of the Ansible code used for the automated setup and configuration of OpenVPN. This will provide a comprehensive understanding of each action performed during the integration process.

- **Variables:**
  - *ansible_python_interpreter*: */usr/bin/python3*
  - *openvpn_admin_username*: "openvpn"
  - *openvpn_admin_password*: "AInRiA@2025!"
  - *openvpn_user1_username*: "work"
  - *openvpn_user1_password*: "IrisA@2025!"
  - *openvpn_user1_ip*: "10.10.10.10"
  - *openvpn_user2_username*: "works"
  - *openvpn_user2_password*: "IrisA@2025!"
  - *openvpn_user2_ip*: "10.10.10.20"
- **Install OpenVPN:**
  - Update system packages.
  - Install required packages, including ca-certificates, wget, curl, net-tools, and gnupg.
  - Add OpenVPN repository key and repository.
  - Update package lists after adding the new repository.
  - Install OpenVPN.
- **Configure OpenVPN Settings:**
  - Set VPN server group pool to "172.0.0.0/8."
  - Configure the number of TCP and UDP daemons to 4 each.
  - Set static network to "10.10.0.0" with a netmask of "16."
- **Restart OpenVPN Access Server:**
  - Restart the OpenVPN Access Server to apply the configuration changes.
- **Configure VPN Client Settings:**
  - Set client Internet traffic rerouting through VPN to false.
  - Do not alter clients' DNS server settings.
  - Restart OpenVPN Access Server to apply changes.
- **Configure User Settings (User 1):**
  - Set autologin for User 1.
  - Change the password for User 1.
  - Set connection IP for User 1.
  - Restart OpenVPN Access Server to apply changes.

- **Configure User Settings (User 2):**
  - Set autologin for User 2.
  - Change the password for User 2.
  - Set connection IP for User 2.
  - Restart OpenVPN Access Server to apply changes.
- **Configure Admin User Settings:**
  - Set the user as a superuser.
  - Set user authentication type to local.
  - Change the password for the OpenVPN admin user.
  - Start OpenVPN Access Server.
- **Display Credentials and URLs:**
  - Display OpenVPN admin, User 1, and User 2 credentials.
  - Get the current server's IP address.
  - Display Admin and Client UI URLs.

2.13.5 Illustration: Integration of OpenVPN Dashboard within the IRISA Platform

This figure presents the admin dashboard of the OpenVPN Access Server, which can be accessed via the following URL: stagepro.tk:943/admin. Notably, two usernames, *work* and *works*, have been pre-configured using Ansible and are prepared for use. This admin dashboard is designed for developer utilization.

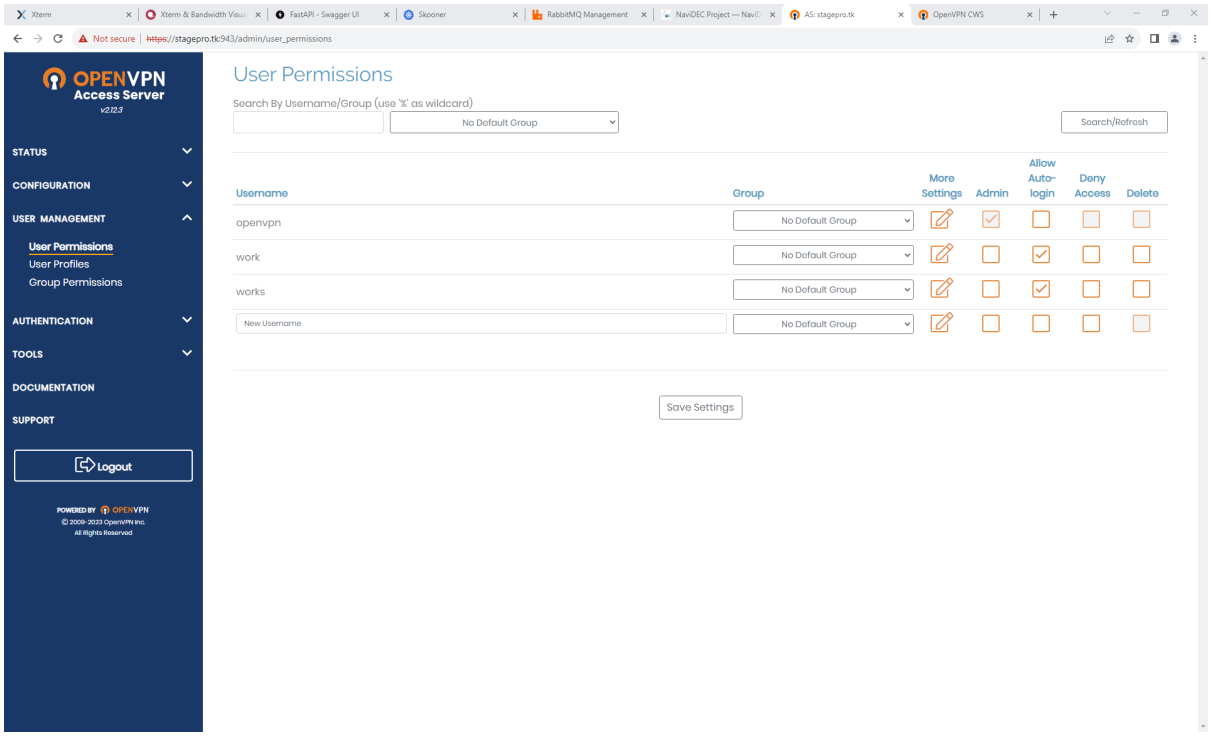

Fig. 21: Display of OpenVPN Access Server for admin Dashboard.

This figure showcases the *work* user dashboard of the OpenVPN Access Server, accessible via the URL: stagepro.tk:943. Users can download the desired OpenVPN operating system. Following this, they can download and import the auto-login profile to OpenVPN and execute it to activate the connection.

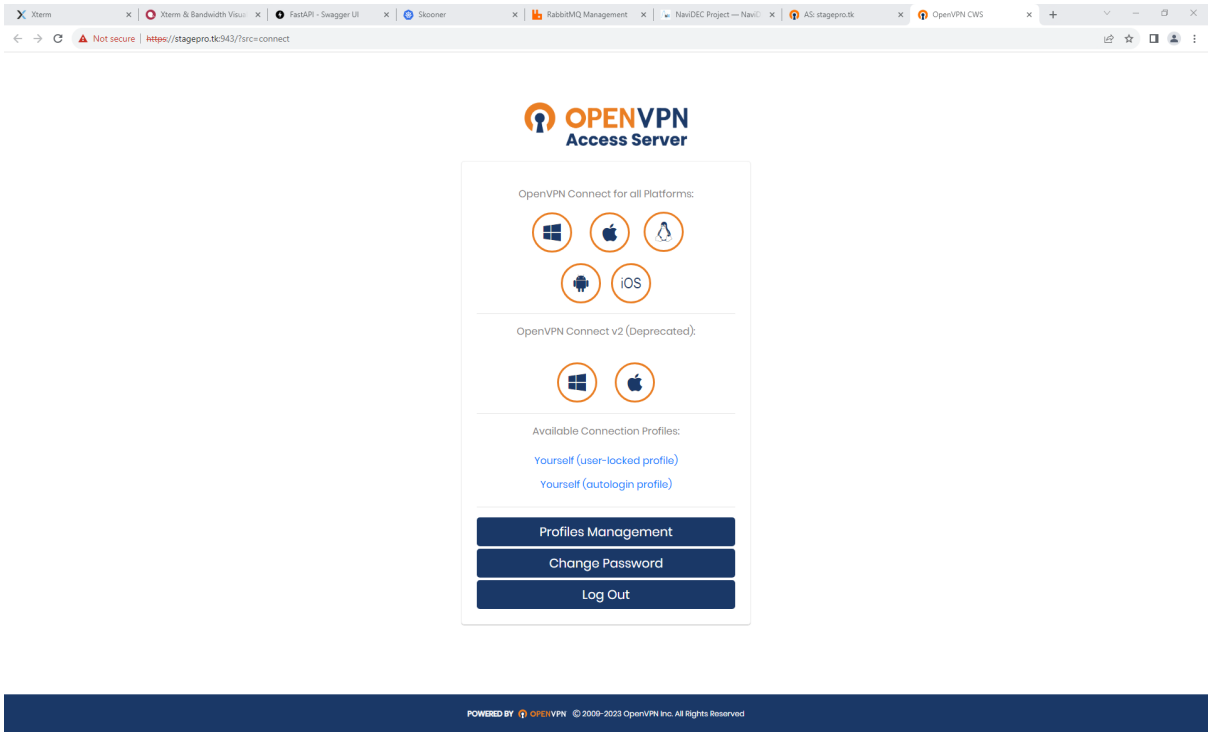

Fig. 22: Display of OpenVPN *work* user dashboard.

This figure demonstrates the successful connection of the OpenVPN Connect Windows application to the *stagepro.tk* server using the *work* user profile.

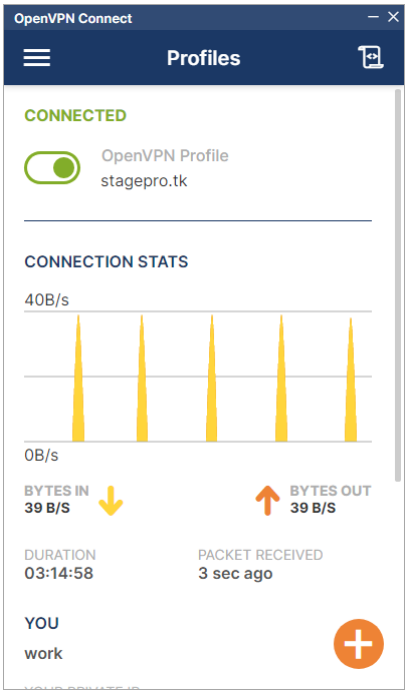

Fig. 23: Display of successful connection of the OpenVPN Connect.

The subsequent figure illustrates the details of the *work* user profile within the OpenVPN Connect Windows application. It includes information such as the private IP address, which is *10.10.10.10*, along with the server IP address, the VPN protocol, and the port used.

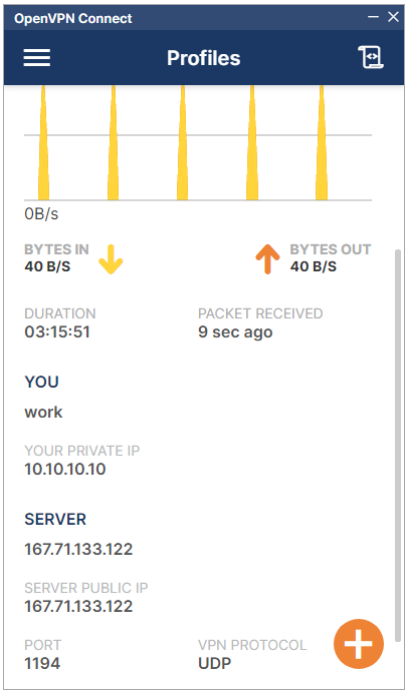

Fig. 24: Presentation of *work* user profile information within the OpenVPN Connect Windows application.

2.13.6 In summery

OpenVPN ensures seamless integration within the IRISA platform, offering secure connections and streamlined user access.

2.13.7 Code Repository

For more details, Click here to view the OpenVPN integration basecode.

2.14 RPC Server Web Browser Launcher

2.14.1 Introduction: Revolutionizing Remote Control with RPC Server Web Browser Launcher

The RPC Server Web Browser Launcher stands as a pivotal component within the IRISA platform, designed to streamline and enhance the user experience. This innovative tool plays a crucial role in automating the launch of web browsers on personal computers or laptops, fostering seamless interaction with the resources available on the IRISA platform.

2.14.2 Why RPC Server Web Browser Launcher within IRISA Platform?

OpenVPN Integration

As discussed in the previous section, the integration of OpenVPN has enabled the remote Ubuntu 20.xx server of the IRISA platform to ping the destination private IP on a PC or laptop, using a private IP, specifically 10.10.10.10.

The Requirement

The subsequent requirement is to design a mechanism capable of executing a specific function: launching a web browser on the personal computer or laptop that is used to access the URL resources of the IRISA platform.

The Solution: RPC Server Web Browser Launcher

This is where the RPC Server Web Browser Launcher tool comes into play. This tool must be operational on the PC or laptop prior to establishing an SSH connection to the remote server of the IRISA platform.

The Process

Upon establishing an SSH connection to the remote server of the IRISA platform and executing the command *./run.sh*, the integrated RPC client within the IRISA platform initiates a verification process to check the reachability of the RPC Server Web Browser Launcher. If it is active, the startup process of the platform proceeds.

The Result

Once the platform is operational, the RPC client sends a request to the RPC Server Web Browser Launcher to launch a local web browser, such as Google Chrome for Windows or Firefox for Linux, on the designated personal computer or laptop. The web browser automatically opens all necessary tabs corresponding to the URLs used by the IRISA platform. This includes, but is not limited to, the xterm.js interactive terminal and bandwidth visualization.

2.14.3 Benefits of Integrating RPC Server Web Browser Launcher with IRISA Platform

Here are some benefits of using RPC Server Web Browser Launcher with IRISA platform:

1. **Automated Browser Launch:**

   The RPC Server Web Browser Launcher automates the process of launching a web browser on the user's device, enhancing the user experience and efficiency.

2. **Seamless Interaction:**

   It fosters seamless interaction with the resources available on the IRISA platform by automatically opening all necessary tabs corresponding to the URLs used by the platform.

3. **Versatility and Cross-Platform Functionality:**

   The tool is versatile and can work with various web browsers such as Google Chrome for Windows or Firefox for Linux. Its ability to support cross-platform functionality makes it adaptable to different user preferences and system requirements.

4. **Efficient Communication:**

   By verifying the reachability of the RPC Server Web Browser Launcher, it ensures efficient communication between the IRISA platform and the user's device.

5. **Streamlined Processes:**

   The tool streamlines the startup process of the platform, making it more user-friendly and efficient.

6. **Enhanced Security and Specific Functionality:**

   Integration of the RPC Server Web Browser Launcher with IRISA platform enhances security of communication between the platform and user's device, ensuring the tool performs specific, designated functions. This functionality bolsters system security by minimizing potential vulnerabilities.

2.14.4 In-Depth Examination of RPC Server Web Browser Launcher of IRISA Platform

This Python script serves as an RPC (Remote Procedure Call) server for launching web browsers on Windows or Linux platforms. It is intended to be used in conjunction with the IRISA platform. Let's break down the key components and functionality:

- **Imports:**
  - The script imports necessary modules, including *os*, *platform*, *subprocess*, and *SimpleXMLRPCServer*.
- **Configuration:**
  - The IP address and port number are assigned to the variables *ip_address* "0.0.0.0" and *port* "8100" .
- **Color Functions:**
  - Two functions (*change_color* and *display_message*) are defined to facilitate printing colored text to the terminal. These are used for better visibility of messages.
- **Function: `launch_browser(urls)`**
  - This function is responsible for launching web browsers on the user's device based on operating system.
  - For Windows, it uses the *psexec* tool to run the *start chrome --new-window* command, opening a new Chrome window with specified URLs.
  - For Linux, it uses the *firefox* command to open URLs in Firefox.
  - Unsupported operating systems trigger an error message.
- **Function: `is_alive()`**
  - This function returns a string indicating that the RPC server is alive. Registered as an RPC function.
- **Function: `start_server(ip_address, port)`**
  - Sets up and starts the XML-RPC server on the specified IP address and port.
  - Displays a title and a message indicating the requirement for Chrome on Windows or Firefox on Linux, and advises on firewall considerations.
  - Registers the *launch_browser* and *is_alive* functions as RPC methods.
  - Initiates the server to listen for RPC calls indefinitely.
- **Main Execution:**
  - Calls *start_server* with the specified IP address and port to start the RPC server.
- **Usage Notes:**
  - Instructions are provided at the beginning, suggesting that users copy the file and run it in a separate terminal before launching the IRISA platform code.
  - It's assumed that Google Chrome should be installed on Windows or Firefox on Linux.

2.14.5 Illustration: RPC Server Web Browser Launcher

This figure illustrates the RPC Server Web Browser Launcher on windows, ready for incomming request

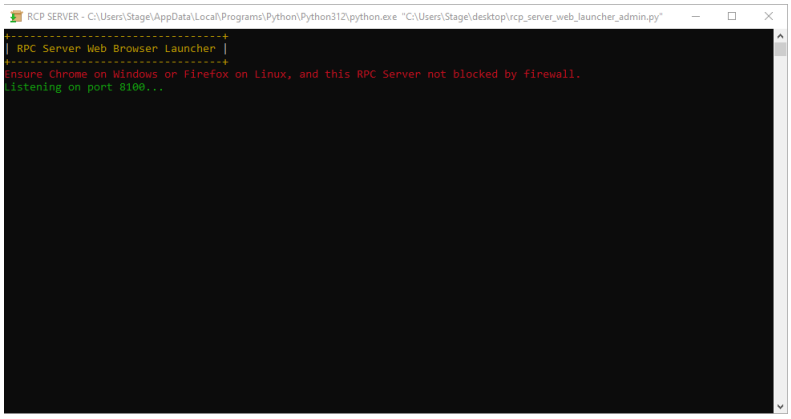

Fig. 25: Display of RPC Server Web Browser Launcher on Windows, prepared to handle incoming requests.

This figure illustrates the RPC Server Web Browser Launcher on Windows as it receives a request.

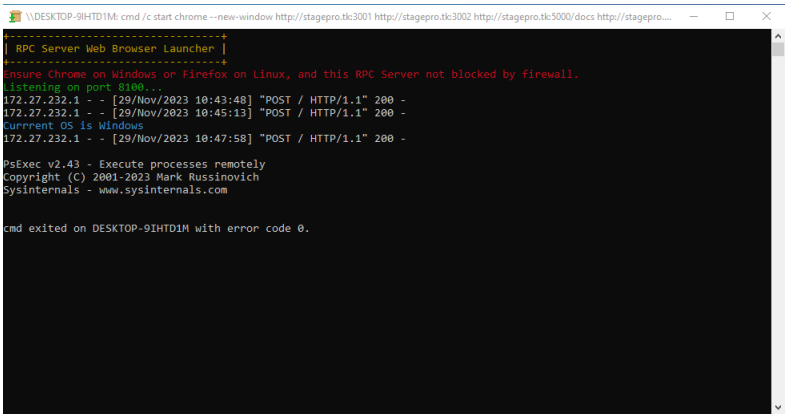

Fig. 26: Display of RPC Server Web Browser Launcher on Windows as it receives a request.

2.14.6 In summary

This script sets up an RPC server to handle requests for launching web browsers. The choice of browser and the URLs to open are determined by the platform's needs. The RPC server is designed to enhance the automation and user experience when interacting with the IRISA platform.

2.14.7 Code Repository

For more details, Click here to view the RPC Server Web Browser Launcher basecode.

2.15 User Guide

2.15.1 Installation

Step 1: Bare-metal installation

Before deploying NaviDec, a prerequisite is to prepare the server. Your server should be running Ubuntu **20.04 LTS**.

First, establish an SSH connection to the remote Ubuntu server and initiate the installation of Ansible on a freshly installed version of Ubuntu 20.xx:

```
sudo apt-get update && sudo apt-get install ansible -y
```

Next, clone the NaviDec repository:

```
git clone -b master https://gitlab.inria.fr/muhammad.al-qabbani/navidec.git
```

Lastly, run the Ansible playbook to install the required dependencies:

```
sudo ansible-playbook -i "localhost," -c local navidec/ansible/configure_server.yml &&
↪ exit
```

Upon completion of the installation, the active terminal will automatically log out. Logging back in will allow for the re-evaluation of Docker group membership.

Step 2: Variables Configuration

The following steps will be follows for preper configuration

1- Modify the credentials_mapping in the web_launcher.py <web_launcher/web_launcher.py>__ file by replacing it with the hostname of the Linux system where the installation was finalized in Step 1 as follows.

A. Substitute the replaceMeWithPropperHostnameOfUbuntuLinuxOSWhereNaviDECIsInstalled variable with the hostname of the Linux system where the installation was completed in Step 1.

B. Set the value of OS_browser to either 'windows' or 'linux', based on the operating system of the local machine that is intended for browser launch.

```
# Modify the credentials_mapping dictionary to store lists of credentials
credentials_mapping = {
    "replaceMeWithPropperHostnameOfUbuntuLinuxOSWhereNaviDECIsInstalled": [
        {
            "OS_browser": "windows",
            "OS_hostname": "10.10.10.10",
        },
        {
            "OS_browser": "linux",
            "OS_hostname": "10.10.10.20",
        },
        {
            "OS_browser": "linux",
            "OS_hostname": "131.254.135.202",
        },
    ],
```

Fig. 27: Display of credentials.

C. Assign the value of OS_hostname to the IP address of the local machine that is intended for browser launch.

D. Should the IP address value assigned to OS_hostname in Step C be a public IP address, and if it's accessible from the remote Ubuntu Linux system to the local operating system, there's no need to install OpenVPN on the local OS. In such a scenario, one can directly proceed to Step 2.

E. If the decision is made to retain the value of OS_hostname as "10.10.10.10", it will be necessary to download the OpenVPN Access Server application that aligns with the local operating system. This can be accomplished via the following webpage: https://YourLinuxUbuntuRemoteServerIp:943. Upon downloading the OpenVPN application that corresponds with the local operating system, secure the 'Yourself (autologin profile)' connection profile and import it into the OpenVPN Access Server application. With these steps completed, a connection can be established.

Use the following OpenVPN client credentials to access the page:

```
OpenVPN URL:    https://yourLinuxUbuntuRemoteServerIp:943
Username:       work
Password:       IrisA@2025!
```

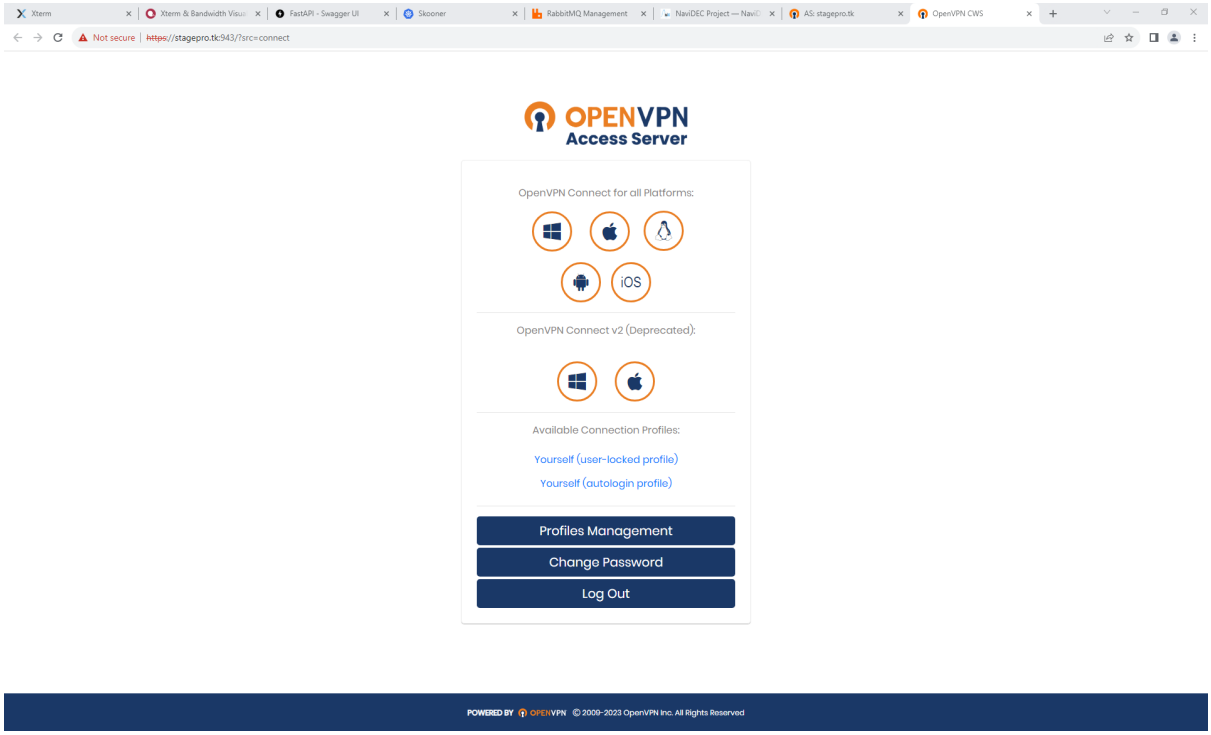

Fig. 28: Display of OpenVPN *work* user dashboard.

2- Initiate the download and launch of the rpc_server_web_launcher.py <web_launcher/rpc_server_web_launcher.py>__ file on the local machine that is intended for browser launch. It's crucial to confirm the pre-installation of Python 3 on this machine. Furthermore, ensure that the file isn't hindered by any active system firewall.

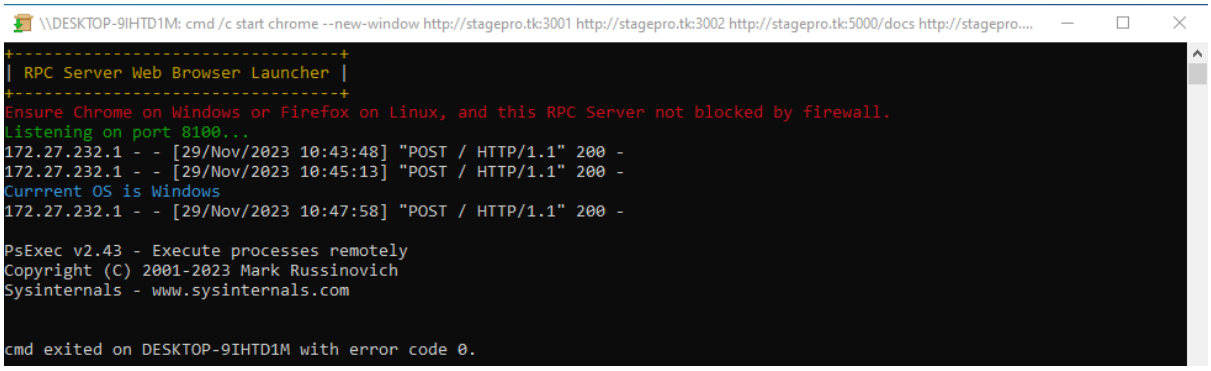

Fig. 29: Display of RPC Server Web Browser Launcher on Windows as it receives a request.

3- Please ensure that Google Chrome is already installed if your local operating system is Windows, and Firefox if your local operating system is Linux.

4- For those using a Windows operating system, it's recommended to execute the following command within an elevated PowerShell environment (Run as Administrator). This step is crucial for enabling the receipt of incoming ICMP echo requests, often known as "pings".

```
# Import the necessary network security module
Import-Module NetSecurity

# Add a new rule to the advanced firewall settings
# This rule allows incoming ICMPv4 echo requests (pings)
netsh advfirewall firewall add rule name="ICMP Allow incoming V4 echo request"␣
↪protocol=icmpv4:8,any dir=in action=allow
```

Please note that this command sequence first imports the required network security module, and then adjusts the advanced firewall settings to permit incoming ICMP echo requests.

5- If your remote Ubuntu server does not have a public domain name, one can edit the hosts file on your local operating system and add the IP address and hostname of the remote Ubuntu server. Here's how one can do it:

```
IPaddressOfUbuntuLinux    hostnameOfUbuntuLinux
```

In this syntax, replace `IPaddressOfUbuntuLinux` with the IP address of your remote Ubuntu server and `hostnameOfUbuntuLinux` with the hostname of your remote Ubuntu server. This will allow your local machine to recognize the remote Ubuntu server by its hostname.

The usual path to the hosts file on Windows is:

```
C:\Windows\System32\drivers\etc\hosts
```

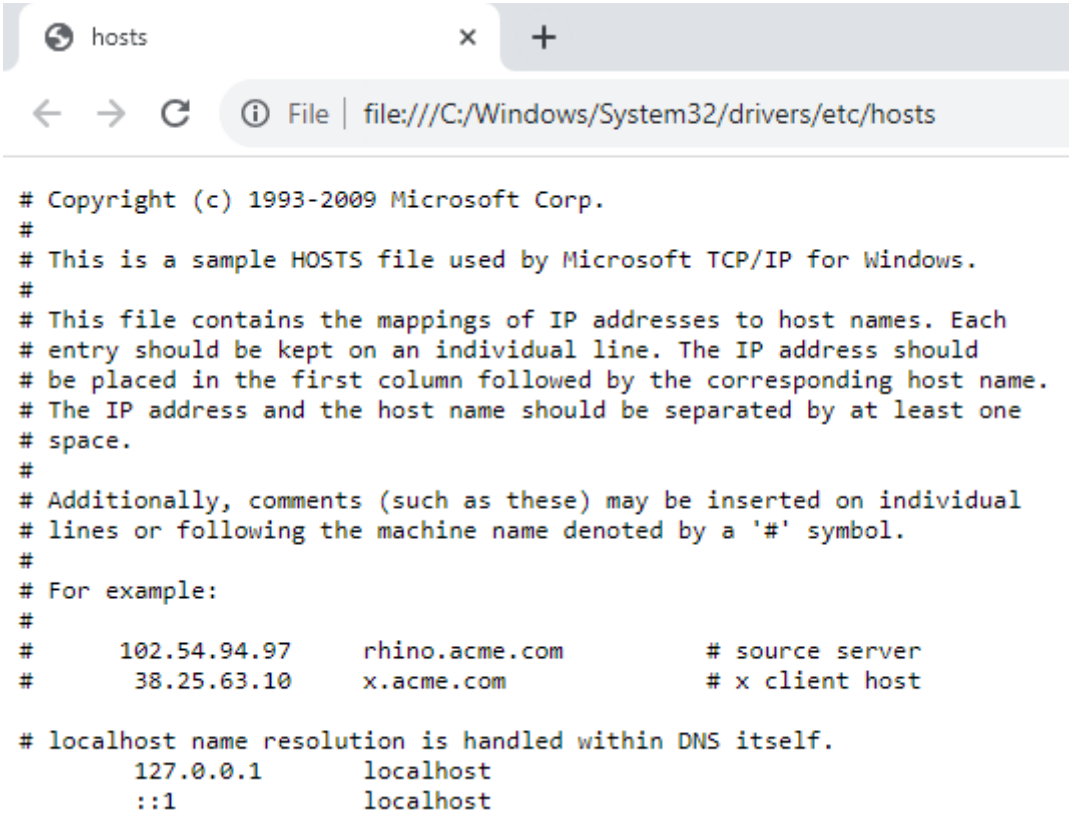

Fig. 30: Display of windows hosts file.

And on Linux, one can find the hosts file at:

```
/etc/hosts
```

```
root@stagepro:~# cat /etc/hosts
# Your system has configured 'manage_etc_hosts' as True.
# As a result, if you wish for changes to this file to persist
# then you will need to either
# a.) make changes to the master file in /etc/cloud/templates/hosts.debian.tmpl
# b.) change or remove the value of 'manage_etc_hosts' in
#     /etc/cloud/cloud.cfg or cloud-config from user-data
#
127.0.1.1 stagepro.tk
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

root@stagepro:~#
```

Fig. 31: Display of linux hosts file.

Please be aware that administrative privileges may be required to modify these files.

Step 3: Runing NaviDEC code

One can now directly start NaviDEC from the remote Ubuntu server via SSH by executing the following commands:

```
./run.sh
```

After initiating the command, kindly allow some time for the application to load. Upon completion of this process, your local browser will launch with multiple tabs. This will facilitate both the viewing and management of the platform.

CHAPTER

THREE

WORK ORGANIZATION METHODOLOGY: A COMPREHENSIVE EXPLORATION OF TECHNICAL GROWTH

3.1 Work Organization Methodology

Here are several key points outlining the approach I employed to structure and manage my work:

1- **Global Task Requirements:**

- **Project Code Comprehension:** Delve into the complexities of the platform code that was handed over to me. This involves understanding its intricacies, refining the codebase, and addressing any deprecated elements.
- **Completion of Essential Development Tasks:** Execute all essential development tasks as outlined in the project requirements. These tasks form the core objectives that need to be accomplished during the course of the project.

2- **Intensive Bootcamp Approach:**

With the significant tasks at hand and only a 5-month timeframe for the internship, I was unsure if this period would be sufficient. Regardless of this uncertainty, I recognized that every moment was precious. This led me to adopt an intensive bootcamp mentality, focusing on stringent time management and disciplined routines to ensure the efficient completion of tasks within the internship period.

3- **Holistic Task Breakdown Approach:**

Embracing my learning journey, which I thoroughly enjoy, I didn't perceive the tasks merely as assignments to be checked off. Instead, I committed myself fully to the project, aiming to produce work of professional quality.

Beyond the essential development tasks outlined in the internship guidelines, I proactively incorporated additional tasks that were introduced during progress reviews with my supervisor.

I also took the initiative to undertake unassigned but crucial tasks. These tasks, though not explicitly required, significantly contributed to enhancing the project's professional presentation.

Regardless of a task's official status, if it had the potential to improve the project, I deemed it important to tackle. All this while, I was acutely aware of the limited timeline of the internship, akin to a ticking clock, reminding me to make the most of every moment.

4- **Task Prioritization:**

Tasks were prioritized based on their nature: Automation, Mandatory, Additional, Complementary.

When I first started on this project, I realized that running the platform required a series of manual commands, including opening multiple xterm consoles one at a time. To gain a better understanding of the full picture and streamline the process, I prioritized automation as my initial task.

Once the automation was in place, I moved on to the mandatory tasks that were required for the project. These tasks formed the core of my responsibilities and were crucial for the progress of the project.

In between meetings with my supervisor, I tackled additional tasks. These tasks, provided me with a deeper understanding of the project and helped me enhance the overall quality of my work.

Finally, I focused on complementary tasks. These tasks, while not mandatory or additional, played a significant role in enhancing the project overall and contributed to a more comprehensive and professional outcome.

5- **Using GitLab Plan for Issue Board and Version Control:**

I utilized the TODO and issues board on GitLab as a valuable tool for noting down my ideas, addressing problems, and planning work for a variety of purposes related to enhancing the IRISA platform. I tailored the issue board to fit my needs and workflow. This allowed me to:

- Define the project according to phases. Currently, I'm in the second phase, which is the internship. The third phase will be the integration after the internship.
- Use it for brainstorming ideas.
- Keep track of tasks and monitor their status.

In addition, Git/GitLab was utilized for version control, allowing for effective tracking of changes, creation of branches, and efficient code management.

This systematic approach to task management not only played a crucial role in ensuring the successful completion of the project within the given timeframe, but it also led to further enhancements and evolution of the project due to the centralization of ideas and tasks in one place. This method proved to be highly effective in managing and advancing the project.

6- **Research and Learning:**

Begin by studying minimalist examples related to the project, adapt them to the given scenario, and integrate them into the platform.

During the execution of my tasks, I adopted a systematic approach. I worked on one task at a time, starting with research to identify the best methods and available open-source tools. I then refined these tools according to the required scenario.

For instance, when I needed to implement a task like putting multiple interactive terminals in a web workspace, I started with a minimalist isolated scenario example. Once it worked and I understood how to set up and operate this new tool, I began to adapt this example step by step to serve the project task.

The next phase involved integrating with the web browser and testing the user experience from an end-user perspective. Following this, I integrated it with the project as a backend, like JavaScript with Python, and finally set up sockets and automation commands.

This approach allowed me to progressively build my understanding and skills while ensuring that each task was completed effectively.

7- **Regular Testing:**

After integrating a new tool into the project or developing one, I conduct thorough testing to ensure everything works as expected. This involves adjusting any undesired outcomes and rectifying any anomalies. This regular testing process is crucial to maintaining the stability of all implemented features in the platform and ensuring the overall quality of the project.

8- **Continuous Code Enhancement:**

The process of continuous code enhancement initiates with the integration and testing of the code with the platform. As new tasks emerge, the code is consistently enhanced. This process leads to numerous enhancements in certain areas of the code, while other parts become deprecated. These deprecated sections are then replaced with more efficient and secure code due to the application of superior, enhanced approaches. A prime example of this is the improved security measure where an RPC server is utilized for browser opening, replacing the provision of SSH credentials, which could potentially be a significant security vulnerability. This continuous enhancement process cultivates an evolving experience and ensures the codebase is consistently up-to-date.

9- **Seeking Assistance:**

Despite my shy personality, I don't hesitate to seek help from colleagues when I encounter a roadblock or a challenging point in the project. I resort to this option only as a last measure when I'm saturated with ideas and need some hints. Their insights and experiences often provide valuable guidance and help me overcome obstacles, ensuring the smooth progression of the project. That's why I've listed this point at the end, as it's a final resort but an essential part of the process.

This methodology provides a structured approach to managing the project, ensuring that all tasks are completed efficiently and effectively. It also emphasizes continuous learning and adaptation, which are crucial for navigating the complexities of the project.

3.2 Education Journey: IRISA Platform - A Path of Technical Growth

3.2.1 Introduction

Embarking on a five-month internship at IRISA, my familiarity with Python was largely theoretical. However, this transformative period wasn't merely a sequence of tasks; it was a dynamic exploration that allowed me to not only apply my existing knowledge but to continuously learn from obstacles and challenges.

3.2.2 Contextualizing the Journey

The initiation of the internship marked the beginning of an immersive experience within the dynamic realm of infrastructure development. The IRISA Platform provided a real-world canvas for learning, experimentation, and innovation. This contextual backdrop sets the stage for understanding the intricacies of the tasks undertaken and the skills acquired.

3.2.3 Navigating Complexity

The IRISA Platform was not merely a collection of isolated tasks; it represented a complex, interwoven ecosystem of technologies. Each of the thirteen tasks, which included understanding code, became a navigational point in this intricate web. These tasks guided the development process through the challenges of integrating diverse tools and frameworks. From the foundational infrastructure-as-code principles to the nuances of user interface design, the journey involved not just technical proficiency but a strategic approach to problem-solving.

3.2.4 A Learning Journey: Embracing Challenges and Opportunities

The internship was a transformative experience, presenting numerous challenges that served as unique learning opportunities.

- **The Power of Automation:**

  An enticing word for me throughout this journey has been "Automation." I found joy in automating every aspect of the project, from infrastructure to document building and deploying on webpages. This emphasis on automation not only streamlined processes but also underscored the significance of efficiency in project development.

- **Enhancing Skills with Python and Node.js**

  Developing a multi-subproject web interface using Python and Node.js marked a significant enhancement in my technical skills.

- **Mastering Containerization:**

  The complexities of containerization with Docker and Kubernetes were unraveled, adding a key aspect to my learning process.

- **Adapting Tools for Creativity:**

  The adaptation of tools like Apache Chart, noUiSlider, and xterm.js to meet project requirements demonstrated a fusion of creativity and technical skill.

- **Security Enhancement through Technology:**

  The application of OpenVPN integration and RPC technology for automated browser launching expanded my experience and led to my involvement in enhancing security.

- **Migration and Organization:**

  The transition from Flask to FastAPI broadened my experience, while the use of Ansible for role organization transformed theoretical knowledge into successful practical application.

- **Project Comprehension through Flowchart:**

  The use of 'flowchart as code' from eraser.io significantly enhanced my understanding of the project as a whole, facilitating my technical growth and adaptability.

- **Navigating Documentation:**

  Despite initial unfamiliarity, I managed to grasp the essentials of Sphinx documentation, aiding in the achievement of project goals.

- **Image Editing:**

  Beyond coding, I ventured into image editing, employing professional tools such as Adobe Illustrator and Adobe Photoshop. This not only enhanced the visual appeal of the project but also emphasized the importance of a polished and professional presentation.

In the realm of technology, every challenge became an opportunity for growth. Each new tool or framework served as a brushstroke on the canvas of my technical proficiency. This report encapsulates the diverse skills acquired and challenges conquered during this enriching period at IRISA, serving as a testament to the transformative nature of this internship.

3.2.5 Collaboration and Knowledge Exchange

The IRISA Platform was not a solo endeavor but a collaborative effort within the IRISA team. Engaging in remote knowledge exchange sessions with project partners, collaborating on intricate components with IRISA colleagues, and seeking guidance from experienced mentors fostered a culture of shared learning. The platform became a nexus for collaborative problem-solving, emphasizing the importance of effective teamwork in navigating complex projects.

3.2.6 Adaptability, Agile Development, and Task Management

The internship was marked by the iterative nature of tasks, from migrating frameworks to integrating third-party tools. This underscored the importance of flexibility and responsiveness in the face of evolving project dynamics. As these dynamics unfolded, the ability to adapt to evolving requirements emerged as a crucial skill.

Building on this adaptability, the application of agile development methodologies became a key focus. This included the use of an issue board integrated within GitLab for task management and scheduling. These methodologies not only enhanced the project's flexibility and responsiveness but also improved task management, emerging as crucial skills throughout the internship.

3.2.7 Beyond Coding: Embracing User-Centric Development

The creation of the Interactive Web-based Terminal and Diagram Illustration system went beyond the realm of traditional coding, venturing into the sphere of user-centric development. The emphasis shifted from producing lines of code to enhancing user experience, prioritizing the development of interfaces that were not only functional but also intuitively designed. This paradigm shift expanded the understanding of technology's impact on end-users. Two prime examples of this approach are the sub-projects involving xterm.js and bandwidth visualization, which effectively illustrate the focus on user-centric design.

3.2.8 Conclusion: A Holistic Growth Experience

To conclude, my journey within the IRISA Platform transcended beyond a mere technical exploration; it was a comprehensive growth experience. Beyond the realm of code and configuration files, it embraced collaboration, adaptability, user-centric development, and a profound understanding of the complex interplay between diverse technologies.

Moreover, my internship at IRISA served as my initial exposure into a French professional environment. This experience not only amplified my technical acumen but also provided me with an authentic exposure to a French workplace. This significantly enhanced my command over the French language, fostered my personal development, and facilitated a comprehensive assimilation of the French mindset.

Fundamentally, the amalgamation of theoretical insights from the first semester at the university and the practical experience from the second semester in a real-world setting has prepared me thoroughly. This all-encompassing preparation has instilled in me the confidence and advanced knowledge necessary to venture into the job market, or "le marché du travail" as it's known in French, positioning me as a strong contender, well qualified in the French job market.

This report stands as a testament to the transformative nature of the internship, revealing the multifaceted aspects of skills honed and challenges overcome during this enriching tenure at IRISA. It encapsulates the essence of my journey, viewing each challenge as a stepping stone towards growth and each new tool or framework as a brushstroke on the canvas of my technical expertise.

3.2.9 Future Prospects: Continuing the Journey

As the internship concludes, it marks not an end, but the beginning of a new chapter. I am thrilled to announce that I will continue my journey with the NaviDEC project in the role of a Design Engineer during Phase 3. This phase will focus on the actual integration of the IRISA platform with our project partners.

The upcoming phase promises to be an exciting and challenging endeavor as we move from the development environment to testing the platform in the real world, specifically in the maritime domain. This will provide an opportunity to validate the robustness of the platform and its capabilities in a dynamic and complex environment.

I look forward to this next phase, ready to apply the skills and knowledge I have gained during my internship, and eager to continue contributing to the success of the NaviDEC project. The journey continues, and I am excited to see where it leads...

3.3 Summary

This report documents the work carried out during a five-month internship at IRISA, focusing on the development of components for a composite service orchestration platform in the maritime domain. The report details the tasks accomplished, the challenges faced, and the solutions implemented.

During the internship, I accomplished 12 tasks, each contributing to the development and enhancement of the IRISA platform. These tasks ranged from infrastructure automation with Ansible Playbook, Docker container automation, to migrating the framework from Flask to FastAPI. Key accomplishments include the development of an interactive web-based terminal using xterm.js, creating bandwidth visualization using Apache ECharts, and the development of a user guide using Sphinx. The user guide was published on a website using GitLab Pages, providing a comprehensive guide for users to navigate and utilize the IRISA platform effectively.

The report is structured into three main chapters. The first chapter provides an overview of the project and the context in which it was carried out. The second chapter delves into the enhanced characteristics of the IRISA platform, detailing the various tasks executed and the challenges overcome. The final chapter explores the work organization methodology, providing a comprehensive account of the technical growth experienced during the internship.

The report concludes with a reflection on the internship experience and the knowledge gained. It also announces the continuation of the journey with the NaviDEC project in the role of a Design Engineer during Phase 3, focusing on the actual integration of the IRISA platform with project partners and testing the platform in the real world, specifically in the maritime domain.

This summary provides a snapshot of the work done during the internship, highlighting the technical skills applied and expanded in a real-world setting. The detailed discussions in the report provide a deeper understanding of each task and the challenges overcome.

3.4 Résumé

Ce rapport documente le travail effectué lors d'un stage de cinq mois à l'IRISA, axé sur le développement de composants d'une plateforme d'orchestration de services composites dans le domaine maritime. Le rapport détaille les tâches accomplies, les défis rencontrés et les solutions mises en œuvre.

Au cours du stage, j'ai accompli 12 tâches, chacune contribuant au développement et à l'amélioration de la plateforme IRISA. Ces tâches allaient de l'automatisation de l'infrastructure avec Ansible Playbook, l'automatisation des conteneurs Docker, à la migration du framework de Flask à FastAPI. Les réalisations clés comprennent le développement d'un terminal web interactif utilisant xterm.js, la création d'une visualisation de la bande passante utilisant Apache ECharts, et le développement d'un guide utilisateur avec Sphinx. Le guide utilisateur a été publié sur un site web utilisant GitLab Pages, offrant un guide complet pour les utilisateurs pour naviguer et utiliser efficacement la plateforme IRISA.

Le rapport est structuré en trois chapitres principaux. Le premier chapitre donne un aperçu du projet et du contexte dans lequel il a été réalisé. Le deuxième chapitre approfondit les caractéristiques améliorées de la plateforme IRISA, détaillant les différentes tâches exécutées et les défis surmontés. Le dernier chapitre explore la méthodologie d'organisation du travail, fournissant un compte rendu complet de la croissance technique vécue pendant le stage.

Le rapport se conclut par une réflexion sur l'expérience du stage et les connaissances acquises. Il annonce également la continuation du voyage avec le projet NaviDEC dans le rôle d'Ingénieur d'études lors de la phase 3, se concentrant sur l'intégration réelle de la plateforme IRISA avec les partenaires du projet et le test de la plateforme dans le monde réel, spécifiquement dans le domaine maritime.

Ce résumé donne un aperçu du travail effectué pendant le stage, mettant en évidence les compétences techniques appliquées et développées dans un contexte réel. Les discussions détaillées dans le rapport fournissent une compréhension plus profonde de chaque tâche et des défis surmontés.

3.5 License

3.5.1 IRISA Platform License

IRISA Platform License is released under the IRISA, Université de Rennes license with the following terms:

Copyright (c) [2023] [IRISA, Université de Rennes]

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of IRISA, Université de Rennes, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL IRISA, Université de Rennes BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

3.6 Disclaimer

3.6.1 Disclaimer

> While every effort has been made to ensure the accuracy and completeness of this document, we acknowledge that errors or omissions may occur. The preparation of this document has benefited from the use of language enhancement tools, including ChatGPT and Bing Chat, to improve grammar, language usage, clarity, structure, coherence, and other aspects of writing. We apologize for any inaccuracies, oversights, or unintended remaining errors, and welcome any corrections or suggestions for improvement.

3.7 Refrences

3.7.1 References and Utilized Tools

[1] Giarré, F., Hadjadj-Aoul, Y., & Aït-Chellouche, S. (2023). A Service Migration Strategy for Resilient Multi-Domain Networks in Outage Scenarios. In 2023 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM) (pp. 1-4). IEEE.

[2] Giarré Federico, 2023 main branch of NaviDEC platform Project repository on gitlab, GitLab Repository

[3] sphinx

[4] Python

[5] Node.js

[6] Skooner

[7] FastAPI

[8] Xterm.js

[9] Aansible

[10] Bing Chat

[11] OpenVPN

[12] Docker Hub

[13] OpenAI Chat

[14] Grafana Loki

[15] Diagrams.net

[16] Stack Overflow

[17] Apache Echarts

[18] Adobe photoshop

[19] Eraser Diagram GPT

[20] GitLab Documentation

[21] DigitalOcean Developer Center