University of Cyprus
Department of Electrical and Computer Engineering

Event-Based Monitoring of Digital Infrastructures

Andreas Menelaou

Submitted to the University of Cyprus in partial fulfillment of the requirements

of the Master in Science (MSc) degree in *"Intelligent Critical Infrastructure Systems"*

Master of Science in
INTELLIGENT CRITICAL
INFRASTRUCTURE SYSTEMS

Department of Electrical and Computer Engineering

University of Cyprus

December 2023

Event-Based Monitoring of Digital Infrastructures

Andreas Menelaou

Thesis Examination Committee

- Prof. Georgios Ellinas (Supervisor)
- Assistant Professor Panayiotis Kolios (co-Supervisor)
- Associate Professor Theocharis Theocharides

Abstract

In life, decision-making often involves weighing options against each other. Ideally, we swiftly discern the best choice among them, simplifying our selection process. Unfortunately, though, most of the time, in a complex IT environment of a Critical Infrastructure System that's not the case. Instead, we are facing complex problems, with a large amount of data and multiple and often conflicting criteria. Thus, we end up with a dilemma, as we cannot decide which choice is the right one for the case under consideration. As a result, the need to analyse all possible choices is created, so that we can be sure that we will make the best possible decision to solve the problem at hand.

To cope with the Critical Infrastructure Systems requirements, we had to find ways to efficiently collect and process data in short times (to be able to make comparisons and estimations between different data sets) and interpret them in useful ways, to reach intuitive and substantial conclusions and take appropriate actions.

Monitoring of digital infrastructures stands as a cornerstone in enterprise information technology strategy, overseeing data centers and their components. Administrators continuously track metrics, ensuring that the organization maintains required production levels. Utilizing trends, they validate infrastructure changes long before applications and services face disruptions. Further, real- time alerts empower administrators to promptly address issues that could disrupt business processes.

The aim of the thesis is to enhance the existing Ticketing System for the Cyprus Police, enabling automated ticket creation based on performance monitoring events from software applications within the Central Police Computerization System—a critical part of Cyprus' Digital Infrastructure Systems. This enhancement will empower Cyprus Police information technology operators and first-level supporters to establish monitoring agents for diverse performance indicators. Additionally, it will generate accessible application programming interfaces in the Ticketing System, enabling the system's operators to engage in predictive maintenance activities.

Acknowledgments

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my master's degree!

First and foremost, I owe an immense debt of gratitude to the KIOS team at UCY, especially to my supervisor Prof. Maria K. Michael, Dr. Georgios Ellinas and Dr Panayiotis Kolios whose insight and knowledge into the subject matter steered me through this research. And special thanks to Lilia Georgiou, whose support as part of Administrative-Kios, went above and beyond, enabling my studies to reach new heights. I am genuinely sorry for any additional workload that might have arisen due to my pursuits. A significant debt of thanks goes to Dr. Philippos Isaia and my brother Dr. Charalambos Menelaou of the KIOS Research and Innovation Center of Excellence for all the inputs and invaluable contributions that made my journey through this research and my master's degree not only possible but also immensely rewarding.

The utmost gratitude is reserved for my family, whose unwavering support sustained me throughout these two years of learning. To my children, I apologize for any added moments of grumpiness during the thesis writing process. And to my wife, Demetra, your unwavering support was the pillar that kept me going. Without you, I might have abandoned these studies long ago. Your incredible support means everything to me, and I promise to clear the kitchen table of all papers as I've promised countless times.

Each one of you has played an integral role in this accomplishment, and I am truly grateful for your support, guidance, and understanding throughout this journey.

Finally, I want to express my gratitude to The Cyprus Police HQ IT Department and all my colleagues who took a leap of faith, allowing me to deploy our work to the Production environment of the infrastructure. The dedication of my fellow Operators of the System truly anchored the application and made this achievement possible.

Table of Contents

Motivation

Flow Chart

List of Figures

Motivation

Police Computer Systems continues to grow with a new system deployed nearly every month. In 2020, the Cyprus Police infrastructure was designated as a Critical Infrastructure of the island. Monitoring this infrastructure primarily relies on Oracle Cloud Manager, utilizing embedded scripts to oversee databases performance and execute essential management tasks. However, questions arise regarding the oversight of Application Servers, Networking, Communication Servers, and hang TCP connections between Application Servers and databases. What about load balancing infrastructure. Whenever an application server encounters stack threats or databases face hanging connections—without a clear understanding of the underlying cause—the current approach involves executing a script to restart all server services, including the admin console.

This context sets the stage for the Thesis Project.

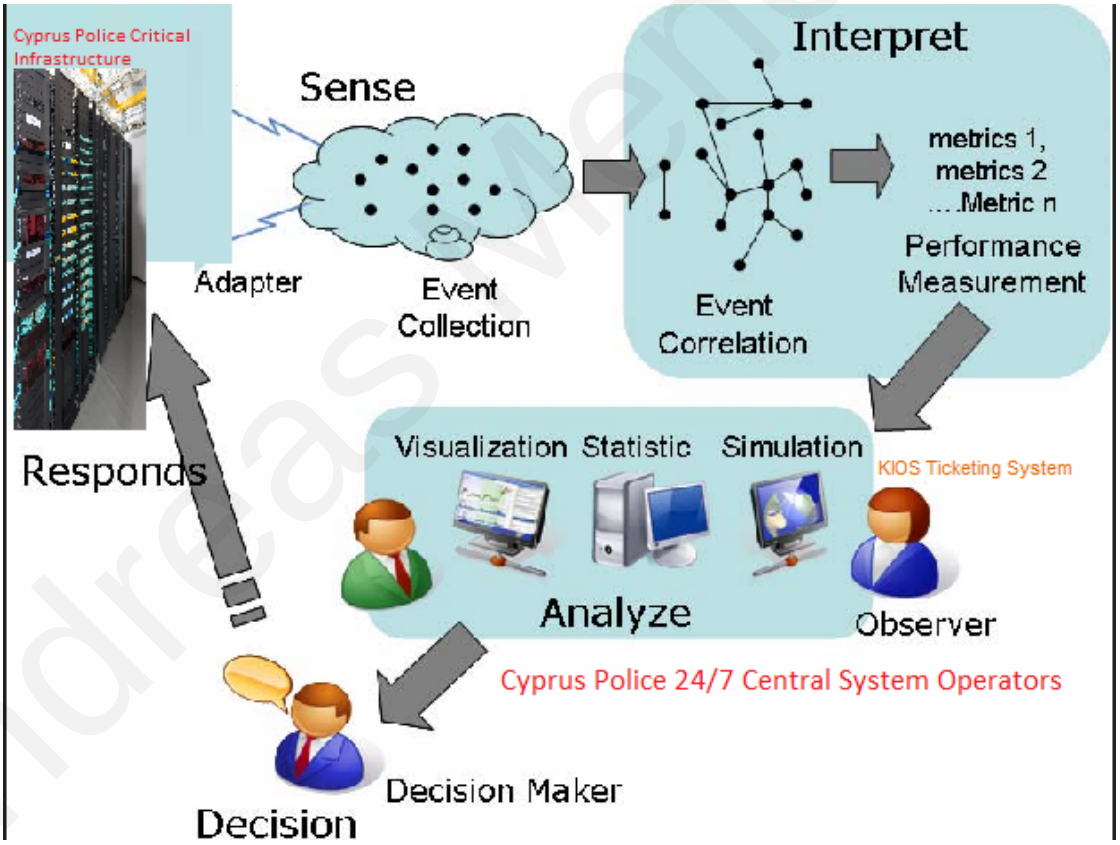


Figure 1 Motivation

Flow Chart

1. **Assess Requirements**	
	• Gather necessary information and prerequisites for deployment.
2. **Prepare Environment**	
	• Check compatibility and readiness of the IT environment. • Set up required servers, databases (PostgreSQL 14) and dependencies. • Prepare networking (access and ports) • Set up Docker and HAProxy • Setup "Nginx" • Setup Celery and Celery-Beat • Setup and Deploy Django • Setup a Python Web Server Gateway Interface-Gunicorn • Setup Docker Compose • Setup and configure Prometheus. • Setup Allert Manager • Setup Node Exporter • Setup and deploy all python scripts to collect data metrics from infrastructure. • Analyze all metrics according to infrastructure with exception rules. • Setup Alerts according to metrics
3. **Configure Application**	
	• Customize application settings • Adjust configurations for database connections, security. • Configure and setup the APIS • programming interfaces in the Ticketing System • Setup monitoring agents for diverse performance indicators
4. **Build Deployment Package**	
	• Compile the Ticket Desk application for deployment.
5. **Deploy Application**	
	• Transfer the deployment package to the target server(s). • Initiate deployment process.
6. **Install Dependencies**	
	• Install any additional software or libraries required by the application.
7. **Configure Database**	
	• Set up the database schema. • Populate initial data
8. **Test Deployment**	
	• Conduct initial tests to ensure the application works in the new environment.
9. **Perform Integration**	
	• Integrate the application with other systems
10. **Finalize Deployment**	

- Double-check configurations and settings.
- Ensure security measures are in place.

11. **User Acceptance Testing (UAT)**
    - Have designated users perform UAT.
12. **Deploy to Production**
    - Once UAT successful, proceed with deployment to the production environment.
13. **Post-Deployment Checks**
    - Verify application functionality post-deployment.
    - Monitor for any issues or errors.
14. **Documentation and Training**
    - Update documentation for the deployed application.
    - Provide necessary training to users/administrators.

This flowchart outlines the sequential steps involved in deploying the Ticket Desk application into Police IT environment.

Chapter 1 - Introduction

1.1 Cyprus Police IT operators Monitoring

"If you don't know where you're going, any road will take you there." George Harrison. That very same lack of insight has plagued IT organizations since the earliest days of computing, leading to inefficiency and wasted capital, nagging performance problems and perplexing availability issues that can be costly and time-consuming to resolve.

Ticketing System uses software-based instrumentation, such as APIs and agents, to gather operational information about hardware and software across the IT infrastructure. Such information can include basic device or application and device health checks, as well as far more detailed metrics that track resource availability and utilization, system and network response times, error rates and alarms, and other data.

IT monitoring employs three fundamental layers. The foundation layer gathers data from the IT environment, often using combinations of agents, logs, APIs or other standardized communication protocols to access data from hardware and software. The raw data is then processed and analyzed through monitoring software. From this, the tools establish trends and generate alarms. The interface layer displays the analyzed data in graphs or charts through a GUI dashboard.

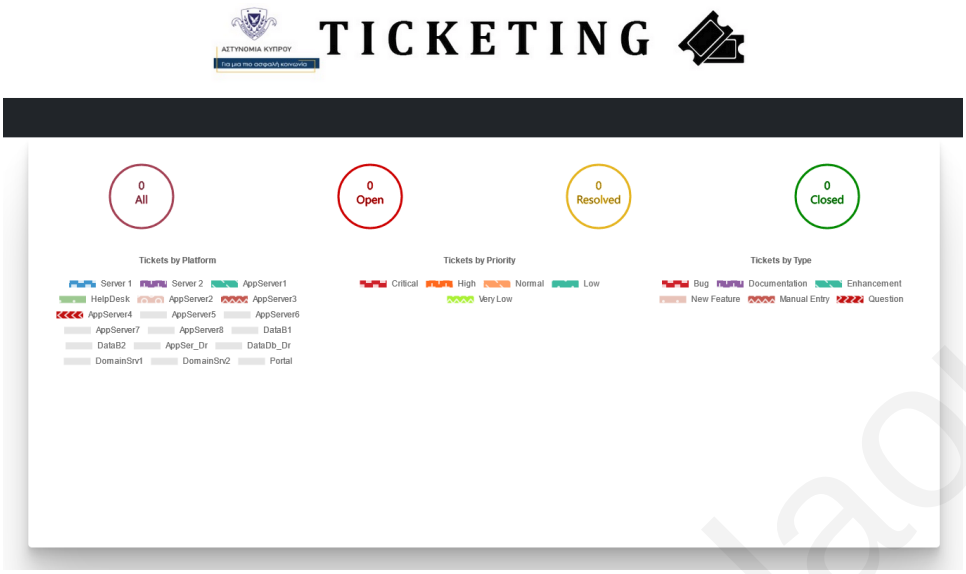


Figure 2 KIOS Ticketing Platform Dashboard

1.2 KIOS Ticketing System

The Kios Ticketing System (Figure , Figure) is designed to efficiently address technical issues and promptly attend to the immediate needs of end users. It primarily operates as a reactive ticket desk, which can be integrated with a service desk operation.

Key functions of the Ticketing System include:

1. Automating Ticket creation, routing, tracking and email notifications. The system streamlines the process of creating tickets, assigning them to the appropriate personnel group, tracking their progress, and notifying relevant parties via email.
2. Acting as a single point of contact for the IT support team. The Kios Ticketing System serves as a centralized hub for Police IT support, allowing users to submit their issues and requests through a single channel.
3. Managing basic service and incident management helps in managing and resolving service-related incidents efficiently. It provides a structures approach to handle incidents, ensuring that they are addressed in timely manner.
4. Displaying self-service options to end-users, enabling them to find solutions to common issues or access relevant information without the need for direct assistance. This empowers users to resolve simple problems on their own, reducing the workload on the support team.
5. Overall, the Kios Ticketing System aims to enhance the efficiency of IT support by automating ticket management, providing a central point of

contact, facilitating incident management, and offering self-service options for end-users.

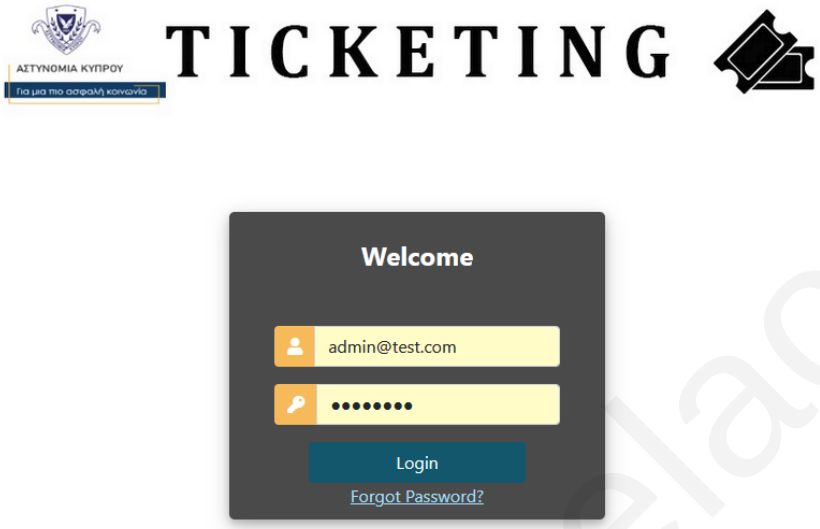


Figure 3 KIOS Ticketing Platform Login

1.3 Ticketing System Architecture

1.3.1 Scaling System resources

Selecting the appropriate hardware equipment for running the application was quite challenge, since the requirements of operational efficiency, experience, and simplify should be in the same line. The equipment requires to operate confidently, tailored to the ICT operator's needs, fast and reliable and furthermore to ensure optimal performance. The hardware architecture of the system setup is based on requirements and scale of the system including the following components:

1.3.2 Application Server resources

The system requires two servers (redundancy) to host the Kios Ticketing application and handle the processing and storage of data. A server with adequate resources ensures excellent performance and user experience. The factors I have considered for estimating the size and type of server, includes the hardware resources such as the processor, memory, and storage as well as other factors like bandwidth, backups, security, uptime and logic to handle peak demands to run efficiently. The load of server is based on the operating system, server functions, software, applications, type and number of files and database. Other factors include the number of users, frequency of access and security tools.

. The CPU handles all processing, calculations, and logic. For reliable and efficient performance, the server requires, fast and powerful processors with the ability to handle and perform several tasks simultaneously. This enables it to perform many computing tasks with high speed and efficiency.

. Memory (RAM) provides a temporary storage for the data that the CPU is processing, and has a better and faster read/write performance than a hard drive. Adequate memory reduces the need for the server to frequently access the slow hard disk memory. A higher amount than the actual requirements have been applied so as to cater for peak demand.

1.3.3 Bandwidth resources

The bandwidth refers to the total amount of data that can transfer to and from Ticketing System. Its size depends on the server use, as well as the frequency and size of clients' requests and the responses they receive.

1.3.4 Network resources

Network Infrastructure is in place to ensure smooth communication between different components. This includes routers, firewalls switches and network cables.

1.3.5 Database Server

The Kios Ticketing System relies on a database Server to store and manage ticket data, user information and other relevant data. For the implementation of this project PostgreSQL is used sitting on SSDs drives to achieve maximum access speed and reliability.

1.3.6 ICT Operators access

End-users access the ticketing system through client devices such as desktop computers or laptops, through the Police closed network. The devices have compatible web browsers to interact with the ticketing system. Also peripheral devices are required, such as printers for generating physical tickets or barcode scanners for ticket validation.

1.3.7 Backup and Redundancy

To ensure data integrity and system availability, backup mechanisms and redundancy measures are be in place. This involves regular backups of the database and redundant hardware components to minimize the risk of data loss or system downtime.

1.4 Central Police Computerized System

The existing system of the Cyprus Police through which the majority of its services are managed is the CPCS (Figure). It was created by a private company and it is managed by in-house servers. CPCS is based on a multi-layered architecture, with a modular design, allowing the addition of new and the extension of existing functions, in a flexible and efficient manner. Specifically, during the operation of CPCS, more than 100 modules have been implemented to serve both operational and administrative needs, and new applications are constantly being added as needed. The CPCS primarily utilizes Java technologies on Oracle WebLogic Server servers for the implementation of business logic of applications, while Oracle Database management software is used for data management. It is fully web-based, and the system's web interfaces are based on Java Server Pages (JSPs), JavaScript, and HTML/CSS.

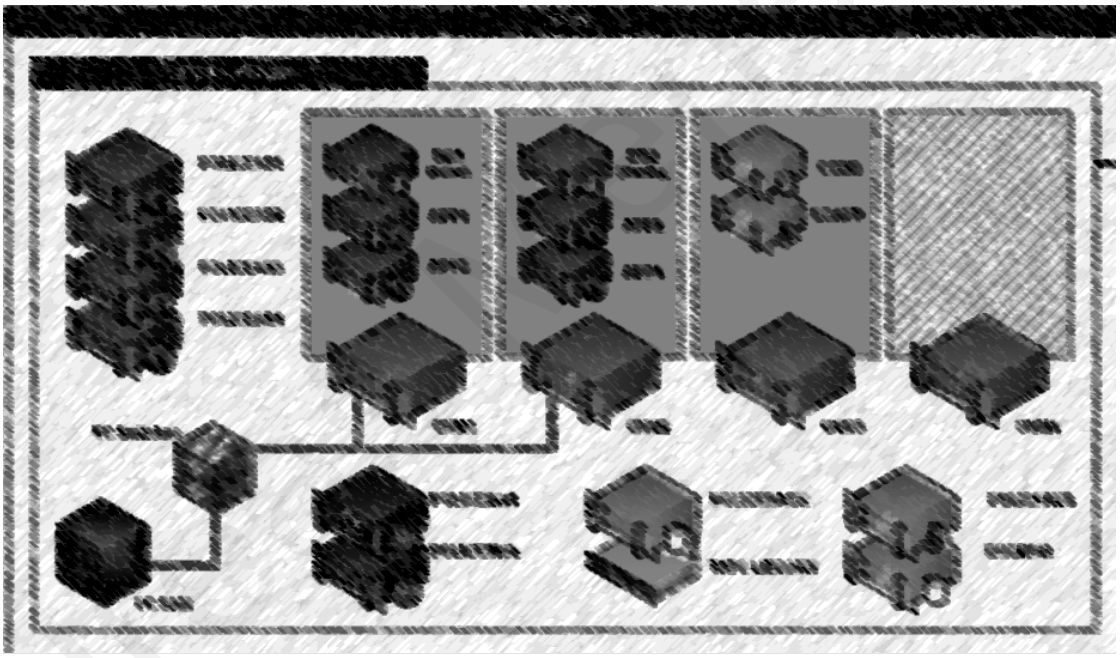


Figure 4 Police CPCS System

Chapter 2 - Literature Review

2.1 Introduction to Server Monitoring

Server monitoring[1] forms a crucial part of IT infrastructure management, ensuring the seamless functioning of servers that are the backbone of modern digital operations. In its essence, server monitoring involves tracking and analyzing server resources to ensure optimal performance and quick responsiveness to any potential issues[2]. The primary goal of server monitoring is to proactively detect and resolve server-related problems, minimizing downtime and ensuring consistent service availability.

In the realm of IT infrastructure, servers play a pivotal role in hosting applications, managing databases, and facilitating communication and data exchange. As such, their health and performance directly impact the overall efficiency and effectiveness of business operations. Server monitoring encompasses a wide range of activities including, but not limited to, tracking server health metrics like CPU usage, memory utilization, disk space, network performance, and application processes. Efficient server monitoring strategies employ various tools and techniques to continuously observe these metrics and generate alerts or reports based on predefined thresholds or detected anomalies. The implications of server monitoring are profound in the context of IT infrastructure. It allows IT administrators and engineers to make informed decisions about resource allocation, load balancing, and system scalability. In cases of server failure or performance degradation, monitoring tools provide critical insights that guide quick and effective troubleshooting. This proactive approach to managing server health not only optimizes resource utilization but also significantly reduces the risk of unplanned outages and service disruptions, which can have dire consequences on business continuity and reputation.

2.2 Evolution of Server Monitoring Tools and Methodologies

The evolution of server monitoring tools and methodologies has been a journey marked by technological advancements and changing industry needs. Initially, server monitoring was a predominantly manual process, involving periodic checks and basic scripting to track system health. This approach was time-consuming, error-prone, and inefficient, especially as IT infrastructures began to grow in complexity and scale.

The advent of automated monitoring tools marked a significant shift in server monitoring practices. These tools allowed for continuous monitoring of server metrics,

generating real-time alerts and detailed reports. As enterprises expanded and the demand for high availability escalated, the need for more sophisticated monitoring solutions became apparent. This period saw the introduction of centralized monitoring solutions that could oversee multiple servers across different locations, offering a comprehensive view of the entire IT infrastructure. The proliferation of cloud computing and virtualization technologies further revolutionized server monitoring. Monitoring tools evolved to accommodate dynamic and distributed environments, where servers could be virtual, cloud-based, or a mix of both. Modern server monitoring tools are equipped with advanced capabilities like predictive analytics, machine learning algorithms, and integrations with other IT management systems. These tools not only track traditional metrics but also analyze patterns, predict potential issues, and automate responses to certain types of incidents.

Current trends in server monitoring emphasize flexibility, scalability, and integration. Tools are now more adaptable, capable of monitoring a vast array of environments and providing integrations with a variety of platforms. The focus has also shifted towards holistic monitoring, where server health is viewed in the context of the entire IT ecosystem, including applications, databases, and network infrastructure.

2.3 Significance of Server Monitoring

Server monitoring is pivotal in maintaining the health and performance of IT systems. In today's digital landscape, where businesses rely heavily on technology for operations, the stability and efficiency of servers are non-negotiable aspects. Effective server monitoring enables organizations to detect and address issues before they escalate into critical problems, thus maintaining high availability and performance standards. The significance of server monitoring extends beyond just problem detection and resolution. It plays a key role in capacity planning and resource optimization. By analyzing trends and usage patterns, organizations can make data-driven decisions about server upgrades, resource allocation, and infrastructure expansion. This proactive planning is essential for maintaining optimal performance levels and ensuring that the IT infrastructure can adequately support business growth and evolving technology demands.

Furthermore, server monitoring contributes to enhanced security. By continuously tracking server activity and system logs, monitoring tools can help identify security threats, unauthorized access attempts, and potential breaches. In an era where cybersecurity threats are rampant and constantly evolving, server monitoring serves as a first line of defense in protecting sensitive data and IT assets. In summary, server

monitoring is a vital component in the management of IT infrastructure. Its evolution from basic manual tracking to advanced, automated solutions reflects the growing complexity and criticality of servers in organizational operations. Effective server monitoring ensures not just the smooth functioning of servers but also supports overall business continuity, security, and growth. As technology continues to advance and IT environments become more intricate, the role of server monitoring in maintaining system health and performance will only become more integral to organizational success.

2.4 Key Metrics in Server Monitoring

Server monitoring encompasses various metrics, each offering insights into the health and performance of server systems. Understanding and effectively utilizing these metrics is essential for maintaining optimal server functionality, ensuring high availability, and preemptively addressing potential issues that may disrupt IT operations.

2.4.1 CPU Usage

One of the primary metrics in server monitoring is CPU usage[3], which indicates the percentage of the processor's capacity currently being used. High CPU usage can lead to slower performance and, in severe cases, can cause the server to crash. Monitoring CPU usage helps in identifying processes that consume excessive resources. It also aids in load balancing decisions and in scaling resources appropriately. For example, consistently high CPU usage may signal the need for additional processing power or optimization of existing processes.

2.4.2 Memory Utilization

Memory utilization[4] is another critical metric, indicating the amount of RAM in use versus the total available. Memory bottlenecks can significantly degrade the performance of applications running on the server. Monitoring memory usage is crucial for identifying memory leaks or applications that are consuming more memory than anticipated. This metric is also vital for capacity planning, ensuring that sufficient memory is available to handle peak loads.

2.4.3 Disk I/O Operations

Disk Input/Output (I/O) operations reflect the read and write operations to the server's storage system. Monitoring these operations is essential as high disk I/O can indicate inefficient application performance or insufficient disk resources. High disk I/O wait can

lead to increased response times for applications, affecting the user experience. Optimizing disk usage, such as through load balancing or upgrading to faster storage solutions, can be guided effectively by this metric.

2.4.4 Network Activity

Network activity metrics, including bandwidth usage, error rates, and throughput, provide insights into the volume of data being transferred to and from the server. Monitoring network activity is crucial in detecting bottlenecks, potential DDoS attacks, and ensuring the network's capacity is adequate for the server's needs. It also helps in bandwidth allocation and in optimizing network configurations for improved data transfer efficiency.

2.4.5 System Uptime

System uptime, a straightforward yet vital metric, measures the time duration for which the server has been running without interruption. Frequent downtimes can be indicative of underlying hardware issues, inefficient resource allocation, or software errors. Monitoring uptime is essential for ensuring high availability and reliability of server-based services.

2.4.6 Load Average

Load average is a unique metric that provides an average of the system load over a period. Unlike CPU usage, which is instantaneous, load average gives a more smoothed overview of system performance over time. It is particularly useful for Unix-based systems and helps in identifying trends in system load, guiding scaling, and resource allocation decisions.

2.5 Automated Monitoring Systems

It's evident that the landscape of IT infrastructure management has significantly evolved. Central to this evolution is the development of automated monitoring systems. These systems are pivotal in managing the complexity and scale of modern IT infrastructures, ensuring high availability, performance, and security. This thesis explores the intricacies of automated monitoring systems, their components, functionalities, and the impact they have on IT operations.

In the early stages of computer science, monitoring was predominantly manual, involving periodic checks and basic alerting mechanisms. However, the advent of complex, distributed systems necessitated a more sophisticated approach. Automated monitoring systems emerged as a solution, capable of continuously tracking a

multitude of metrics across various components of IT infrastructure. These systems are not just a convenience but a necessity in today's fast-paced, data-driven environment where downtime or performance degradation can have severe implications.

An automated monitoring system typically consists of several key components:

a. **Data Collection Agents**: These are programs or agents installed on servers and devices to collect performance data.
b. **Central Repository**: A database or storage system where collected data is aggregated and stored for analysis.
c. **Analysis Engine**: The core of the system, which processes and analyzes the data, often using sophisticated algorithms to detect patterns or anomalies.
d. **Alerting Mechanism**: A system that notifies administrators or stakeholders when predefined thresholds are breached or anomalies are detected.
e. **Dashboard and Reporting Tools**: User interfaces that provide real-time and historical data insights through various forms of visualization.

2.6 Challenges in Server Monitoring

Server monitoring is critical in ensuring the smooth operation of IT infrastructure, yet it is fraught with challenges that can impede effectiveness. This thesis delves into these challenges, exploring their nuances and offering insights into potential mitigation strategies[5][6][7].

2.6.1 Complexity of Modern IT Environments

The diversity and complexity of contemporary IT environments present a significant challenge in server monitoring. With the advent of cloud computing, virtualization, and distributed architectures, monitoring systems must now track a multitude of dynamic components. The heterogeneity of these environments, often comprising a mix of on-premises, cloud-based, and hybrid systems, adds layers of complexity in data collection and analysis. This diversity demands a monitoring solution that is not only robust but also highly adaptable to varying technologies and architectures.

2.6.2 Data Volume and Management

Server monitoring systems often grapple with the sheer volume of data generated by modern servers. This data deluge, while rich in information, can lead to difficulties in data management and analysis. Sifting through this vast amount of data to extract meaningful insights requires advanced data processing capabilities and can be

resource-intensive. Furthermore, the storage and archival of this data pose additional challenges, particularly in terms of cost and compliance with data retention policies.

2.6.3 Real-Time Analysis and Response

The need for real-time analysis and rapid response is paramount in server monitoring. In high-traffic environments, even minor issues can escalate rapidly, leading to significant service disruptions. The challenge lies in developing monitoring systems that are not only capable of real-time data analysis but also equipped to automate immediate responses to potential threats or performance bottlenecks. Achieving this level of responsiveness requires sophisticated algorithms and integration with automated management tools.

2.6.4 Integration and Compatibility Issues

Integrating monitoring tools with existing IT infrastructure can be a daunting task, especially in environments with legacy systems or diverse technology stacks. Compatibility issues can arise, hindering the seamless operation of monitoring solutions. This challenge is compounded by the continuous evolution of IT technologies, requiring monitoring systems to be regularly updated or replaced to maintain compatibility.

2.6.5 Alert Fatigue and False Positives

Alert fatigue, driven by the overabundance of notifications and false positives, is a significant issue in server monitoring. Excessive alerts can desensitize IT staff, leading to delayed responses to actual critical incidents. The challenge lies in fine-tuning alert mechanisms to ensure that notifications are both accurate and actionable. This requires intelligent threshold setting, anomaly detection, and perhaps the incorporation of AI and machine learning techniques to reduce false positives.

Chapter 3 - Ticketing and Monitoring Systems

The Ticketing System architecture consists of various services including a database, a reverse proxy, a Web server, an in-memory data structure store, an asynchronous task runner as well as several Python Web Server Gateway Interface Hypertext Transfer Protocol (WSGI HTTP) servers.

3.1 Services

As show in Figure 5, the incoming traffic will pass through CY Police Infrastructure before it reaches the server hosting the Ticketing backend.

As soon as the requests reach the Ticketing server, they will be load balanced by

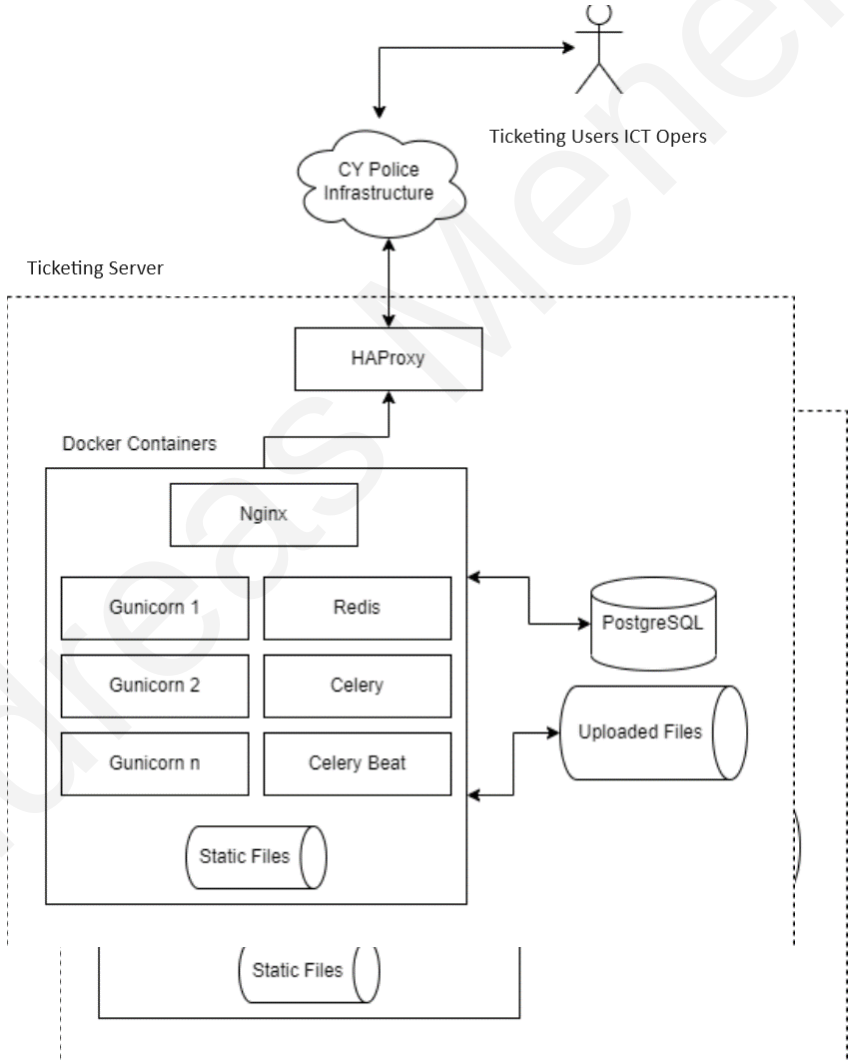


Figure 5 Platform High-Level Architecture

HAProxy to the Ticketing Cluster. To store any information, the Ticketing Docker Cluster uses the PostgreSQL 14 database.

3.1.1 Docker

Docker[8], a prominent tool in software development, is renowned for revolutionizing containerization technology. This technology has been a game-changer in the field, enabling developers to package applications and their dependencies into virtual containers. These containers efficiently run across various computing environments, ensuring consistent application performance irrespective of the operating system and underlying infrastructure differences[9]. The fundamental concept behind Docker is its container-based approach, where containers are lightweight, standalone, executable packages comprising all necessary components to run a piece of software. This includes the code, runtime, system tools, libraries, and settings. This approach is distinct from traditional virtualization methods. Unlike virtual machines that require an entire operating system per instance, Docker containers share the host system's kernel, enhancing resource efficiency and significantly reducing server and IT costs[10].

Docker containers are derived from Docker images, which are essentially container snapshots. These images, stored in Docker registries such as Docker Hub, act as blueprints for creating containers. This concept of Docker images and registries has revolutionized software deployment, simplifying version control, application sharing, and updates. The Dockerfile, a crucial component for building Docker images, facilitates automated and consistent application deployment processes, benefiting continuous integration and continuous delivery (CI/CD) pipelines. A key contribution of Docker is addressing the "it works on my machine" problem. By providing a consistent environment from development to production, Docker ensures uniform software operation across different environments. This consistency is especially beneficial in complex systems requiring varied environments for different components. Docker also excels in networking, offering a powerful and flexible framework to facilitate communication between containers, either on the same host or across different hosts. This capability is essential for microservices architectures, where individual components are independently developed, deployed, and scaled. Docker significantly aids in scalability and orchestration, vital for managing large-scale applications and microservices. Tools like Docker Swarm and Kubernetes handle multiple Docker containers, managing tasks such as load balancing, scaling, and maintaining high availability. These features are crucial in sustaining application performance and reliability under diverse load conditions. In terms of security[11], Docker provides robust isolation, critical in multi-tenant environments, but also introduces challenges in

securing containers and the applications within. Docker continually evolves, introducing new features and best practices for securing containerized environments. Docker's impact extends beyond typical development workflows and IT operations. It fosters a more agile development process with quicker deployment cycles and enhanced team collaboration. In DevOps practices, Docker promotes a culture of rapid feedback loops and efficient problem-solving. Its application in academia and research transcends traditional software development, providing tools for creating reproducible research environments. This reproducibility is crucial in scientific computing, where environmental consistency significantly influences experimental outcomes. Docker's utility in machine learning and data science workflows is notable, offering solutions for packaging and distributing data science applications, maintaining complex dependencies, and ensuring consistency in library and tool versions.

In summary, Docker's multifaceted contribution to the software development world includes simplifying application deployment, ensuring environmental consistency, aiding in microservices architecture, enhancing security, and supporting scalable solutions. Its widespread impact reshapes how developers, researchers, and IT professionals approach software development and deployment. With its growing community and evolving ecosystem, Docker continues to adapt, introducing new tools and features to meet the dynamic demands of technology and development practices.

3.1.2 HAProxy

HAProxy[12], also known as High Availability Proxy, has become an essential component in contemporary IT infrastructure, especially notable for its effectiveness in load balancing and functioning as a proxy server for TCP and HTTP-based applications. The sophisticated functionality and pivotal role of HAProxy in ensuring high availability, reliability, and performance of web services is particularly pertinent. Designed to manage high traffic volumes and provide unwavering availability, HAProxy is a favored solution for businesses where downtime is not an option. It functions by evenly distributing client requests across multiple servers, preventing any single server from becoming a bottleneck[13]. This load balancing is crucial for both maintaining consistent uptime and enhancing user experience as traffic increases. HAProxy is distinguished by its dual capability of performing both layer 4 (transport layer) and layer 7 (application layer) load balancing. Layer 4 balancing is based on network and transport layer data like IP addresses and TCP ports, whereas layer 7 balancing utilizes application layer content, including HTTP headers and SSL session IDs. This dual-layer balancing enables HAProxy to support a diverse array of applications and services, making it adaptable to various environments.

A standout feature of HAProxy is its proficiency in managing thousands of simultaneous connections, a critical attribute in high-traffic scenarios. This, combined with efficient resource utilization, makes HAProxy a cost-effective option for scaling web infrastructures. Additionally, HAProxy enhances the security and reliability of web applications through features like SSL termination, which offloads SSL processing from application servers to improve performance. HAProxy also includes mechanisms for detecting and mitigating several attack types, including DDoS attacks, positioning it as a vital component in a comprehensive web application security strategy. HAProxy's health check functionality is highly acclaimed, as it continuously monitors the status of backend servers, ensuring traffic is directed only to operational servers. This proactive monitoring helps in averting potential downtimes and service disruptions. The logging and monitoring capabilities of HAProxy are also significant, providing detailed insights into traffic patterns, server health, and potential issues, enabling proactive infrastructure management and optimization. The customizability of HAProxy is another key strength. It offers an extensive range of configuration options, making it possible to tailor its performance to meet the specific needs of different applications and environments. This flexibility extends to its integration with other tools and services, enhancing its utility in various scenarios. In cloud computing environments, the role of HAProxy becomes increasingly vital. It is adept at balancing loads across multiple cloud instances, capitalizing on the scalability and redundancy benefits of cloud technology. Its compatibility with containerized environments further adds to its versatility, facilitating efficient traffic management in microservices architectures.

Beyond its core load balancing and web traffic management functions, HAProxy plays a significant role in continuous delivery and deployment within DevOps practices. It enables reliable traffic distribution across multiple servers and environments, supporting various deployment strategies crucial for agile software development, including blue-green deployments and canary releases. For large-scale, distributed systems, HAProxy's session persistence feature, or 'sticky sessions', is essential for maintaining client sessions with specific backend servers, a necessity for applications requiring stateful sessions. This capability, coupled with its proficiency in managing WebSocket traffic, makes HAProxy an excellent choice for real-time applications and services.

3.1.3 PostgreSQL

PostgreSQL[14], commonly known as Postgres, is a highly respected open-source, advanced object-relational database management system (ORDBMS), celebrated for its reliability, robustness, and performance capabilities, particularly in managing

complex data types and large volumes of data[15]. It stands out in the database technology sphere for blending the traditional relational database model with advanced object-oriented database functionalities, making it versatile for a wide spectrum of applications, ranging from small single-machine applications to large Internet-facing applications with numerous concurrent users. At its foundation, PostgreSQL is known for its commitment to standards compliance, being ACID-compliant and largely implementing the SQL:2011 standard. This compliance is vital for ensuring consistent and reliable data handling, crucial for applications demanding high data integrity. PostgreSQL supports various replication techniques, such as synchronous, asynchronous, and logical replication, offering robust solutions for high availability and disaster recovery scenarios. A key feature of PostgreSQL is its support for complex data types like JSON, XML, arrays, and user-defined types, which is particularly beneficial for modern web applications needing diverse data format handling. The JSON support has been a transformative feature, allowing developers to efficiently store JSON documents and conduct complex queries, aligning with NoSQL-like application needs while providing relational database benefits.

PostgreSQL's extensibility is a notable aspect of its design. It permits developers to create custom data types, functions, and write code in various programming languages within the database. This flexibility is invaluable in specialized domains that require customized procedures and operations. The PostGIS extension, supporting geographic objects, further positions PostgreSQL as a preferred choice for geographic information systems (GIS) and location-based services. The database excels in concurrency control and transaction management through its implementation of Multi-Version Concurrency Control (MVCC). This feature gives each user a database "snapshot," enabling concurrent transaction processing without locking conflicts, enhancing performance and scalability, particularly in high-concurrency environments. PostgreSQL's querying capabilities are enhanced by a powerful query planner/optimizer, sophisticated indexing techniques, and a robust query execution engine, making it suitable for data analysis and business intelligence applications. Advanced SQL features like window functions, common table expressions (CTEs), and foreign data wrappers enable data integration from varied sources.

Security is a strong focus for PostgreSQL, supporting numerous authentication methods and robust access control mechanisms to manage data access precisely. Known for its high performance across diverse workloads, PostgreSQL has an efficient query execution engine and features like table partitioning and parallel query execution, enhancing its performance capabilities. The database also allows the creation and use

of indexes on functions and expressions, contributing to quicker data retrieval for complex queries. For developers and database administrators, PostgreSQL offers various tools for database maintenance, including vacuuming and a robust set of backup and recovery tools, ensuring the long-term health and performance of the database. PostgreSQL's vibrant community significantly contributes to its continuous development and improvement, providing extensive documentation, third-party tools, and active forums for support and discussion. This community-driven model ensures PostgreSQL stays at the forefront of evolving user needs. In practical applications, PostgreSQL is used in numerous fields, from web services and e-commerce to financial services and scientific research. Its reliability, advanced feature set, and flexibility make it a suitable choice for organizations seeking a robust, scalable, and feature-rich database management system. Overall, PostgreSQL represents a mature, feature-rich, and highly capable database management system, well-suited for a wide range of applications. Its blend of advanced features, extensibility, robust performance, and strong community support makes it an excellent choice for organizations seeking a reliable, scalable, and flexible database solution.

3.1.4 Nginx

Nginx[16], pronounced as "Engine-X," is a high-performance web server renowned for its stability, rich feature set, simple configuration, and low resource consumption. It was initially designed by Igor Sysoev in 2002 to address the C10K problem, which involves efficiently handling a large number of concurrent connections. Since its first public release in 2004, Nginx has evolved significantly, offering versatile capabilities that make it a favorite in modern web architectures due to its efficiency and speed. Central to Nginx's architecture is its event-driven, asynchronous nature, which allows it to handle concurrent connections with minimal hardware resources[17]. This approach contrasts sharply with the traditional thread-per-connection model used by many web servers, which often becomes inefficient under heavy loads. Nginx's architecture enables it to use memory more efficiently and scalably, particularly beneficial for managing high-traffic websites and resource-intensive back-end applications. As a web server, Nginx excels in serving static content, managing thousands of concurrent connections with low memory overhead. Its role as a reverse proxy allows it to manage and direct incoming traffic to other servers, functioning as a load balancer and enhancing the performance, scalability, and reliability of web applications. This feature is particularly useful in microservices architectures, where Nginx optimizes resource use by balancing traffic across various microservices.

Nginx's functionality extends to performing tasks like SSL termination, offloading the decryption of SSL/TLS traffic from application servers to the Nginx server, which improves overall performance. Its content caching capability further reduces the load on application servers and speeds up response times, with a highly configurable caching mechanism that allows for precise control over cached content. The configurability and flexibility of Nginx are notable. Its straightforward and easy-to-understand configuration files make it suitable for both simple projects and complex enterprise applications. In terms of security, Nginx offers robust features, supporting the latest SSL/TLS protocols and providing mechanisms to limit access to resources and protect against common web vulnerabilities like DDoS attacks. Nginx's design efficiently utilizes system resources, making it well-suited for environments with high demand and large numbers of concurrent users. Its versatility is evident in advanced deployments, where it can serve as a front-end proxy to web application servers like Apache or Tomcat, handling static content and redirecting dynamic content requests.

Moreover, Nginx seamlessly integrates into containerized environments managed by Docker or Kubernetes, enhancing its appeal in modern DevOps practices. This integration allows Nginx to route or balance traffic between different containers, adapting to dynamic environments for optimal resource utilization. Nginx's logging and monitoring capabilities are also crucial, providing detailed access and error logs for troubleshooting and performance tuning. When used with monitoring tools, Nginx offers a clear view of a system's health and performance. For developers and system administrators, Nginx presents a balance of power and usability. Its modular architecture supports the extension of capabilities through third-party modules, and its active community and comprehensive documentation make it accessible for users of various skill levels.

3.1.5 Celery

Celery[18] is a prominent open-source, distributed task queue system, highly regarded in the software development industry for its capability to efficiently handle asynchronous tasks and scheduling. This system is especially critical in modern web development for environments where real-time processing is essential. For an experienced computer scientist in ICT and software development, Celery's intricate architecture and wide application scope are of significant professional interest. Fundamentally, Celery operates asynchronously, managing tasks outside the main program flow, which optimizes resource usage and enhances user experience by allocating long-running tasks to separate processes or machines. It's frequently utilized in web applications for executing background tasks like sending emails, processing

data, or conducting computationally intensive tasks, which, if run synchronously, could hinder application performance. Celery's architecture, characterized by its simplicity and power, employs a distributed messaging system for sending tasks to worker nodes for execution. This scalable and flexible architecture enables Celery to manage a diverse range of tasks, from straightforward functions to complex workflows. Task communication is facilitated by a message broker, with RabbitMQ and Redis being popular choices, ensuring reliable and efficient message delivery to workers.

Scalability is a cornerstone of Celery's design, allowing it to concurrently handle numerous tasks, making it suitable for high-load systems. This scalability is achieved by distributing tasks across multiple workers and machines, catering to fluctuating workloads without necessitating significant infrastructural changes. Additionally, Celery's robust support for scheduling enables tasks to be executed at specific times or intervals, akin to cron but with the benefits of distributed execution and dynamic runtime scheduling. Fault tolerance in Celery is paramount, featuring mechanisms to prevent task loss in case of worker failures. This aspect is crucial for applications reliant on timely and precise task processing. Celery supports task retries for re-execution in case of failures and acknowledgments to ensure tasks are only removed from the queue after successful execution. Performance-wise, Celery is designed to be efficient and lightweight, capable of handling a high volume of tasks with minimal overhead, thus suitable for high-performance, low-latency applications. This efficiency is achieved through event-driven concurrency, enabling concurrent handling of numerous tasks without the need for extensive threading or processes.

Monitoring is another critical feature of Celery, with tools like Flower providing real-time insights into task progress, worker status, and other vital metrics. This monitoring is essential for diagnosing issues, optimizing performance, and understanding the behavior of distributed tasks. Celery's customizability offers a broad spectrum of configuration options, allowing it to be tailored to specific application needs, whether in task routing, defining retry policies, or configuring the message broker. In practical use, Celery is employed across various sectors, from e-commerce and social media to financial services and scientific computing, demonstrating its suitability for a wide range of applications, including real-time data processing and batch processing. Its seamless integration with popular web frameworks like Django and Flask enhances its appeal to Python developers, though its functionality is not limited to Python alone, as it can interact with other languages and platforms through message protocols.

3.1.6 Celery-Beat

Celery-Beat[19], an essential component of the Celery task queue system, serves as a sophisticated scheduler that enhances Celery's capabilities by introducing the functionality of timed job processing. This feature is particularly valuable for automating and managing time-based tasks within complex systems. Celery-Beat operates by scheduling tasks that Celery workers then execute, playing a crucial role in managing periodic and scheduled tasks. This system helps streamline routine tasks such as data backups, report generation, or maintenance jobs by automating them according to a predefined schedule. The use of 'clocked' tasks, which are set to run at specific intervals or times, marks Celery-Beat as an invaluable tool for these operations. At the core of Celery-Beat's functionality is its seamless integration with the Celery framework. While Celery focuses on task execution, Celery-Beat serves as the scheduler, ensuring tasks are triggered as per the schedule. This separation allows for a more scalable and maintainable system. Celery-Beat can run either as a standalone service or alongside a Celery worker, offering flexible deployment options based on the architecture and requirements of the application.

One of the most powerful features of Celery-Beat is its support for various scheduling strategies. It accommodates fixed-time scheduling for executing tasks at specific moments, interval-based scheduling for regular task execution, and crontab-like scheduling for more complex, recurring patterns. This versatility makes it suitable for a broad range of applications. Celery-Beat's integration with Django, a widely-used Python web framework, extends its capabilities to managing web operation-related tasks, such as database maintenance and cache invalidation. This integration leverages Django's ORM and admin interface, facilitating task management and reducing the learning curve for developers. Scalability is a key aspect of Celery-Beat, capable of handling an increasing volume of scheduled tasks. Its distributed nature works in tandem with multiple Celery workers across different nodes, ensuring system performance and reliability even under high task volumes. In terms of fault tolerance, Celery-Beat is designed to manage failures gracefully, preserving scheduled tasks in the event of a system crash or network issue through persistent scheduling. This robustness is vital in production environments where continuous operation is paramount.

Monitoring and management capabilities are also significant in Celery-Beat. With integration with tools like Flower, it provides oversight into task scheduling and execution, offering administrators the ability to monitor task queues, view executed task histories, and manage schedules directly via a web interface. Celery-Beat is not

only customizable but also extensible. Developers can define custom scheduling strategies or add new schedulers, enhancing its adaptability to specific application needs. In practical applications, Celery-Beat is utilized across various industries, from e-commerce to finance and healthcare, proving its suitability for a wide array of applications that require precise timing, like transaction processing or real-time analytics.

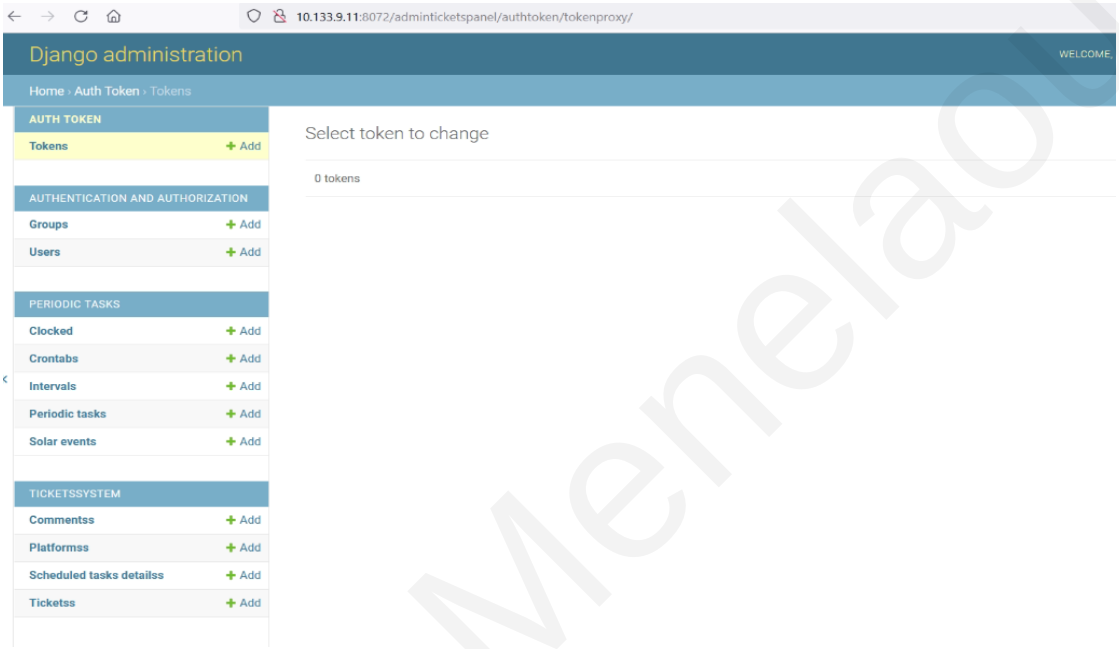


Figure 6 Django Dashboard

3.1.7 Redis

Redis[20], standing for Remote Dictionary Server, is a renowned open-source, in-memory data structure store, widely used as a database, cache, and message broker. Its exceptional speed and efficiency make it a favored choice among developers, particularly for applications requiring rapid data read/write access, such as gaming, high-speed transactional systems, or real-time analytics. Redis's primary feature is its in-memory data storage, enabling significantly faster data access compared to disk-based storage systems. This capability is essential for applications that process large volumes of requests per second. Unlike traditional databases, Redis's in-memory nature allows for extraordinarily quick read and write operations. Despite being primarily an in-memory store, Redis ensures data durability and persistence through mechanisms like snapshotting and append-only files (AOF), which enable it to recover its state after restarts or failures. Redis supports a broad range of data structures, including strings, lists, sets, sorted sets, hashes, bitmaps, hyperloglogs, and geospatial

indexes. This versatility allows for varied use cases, such as using lists for queues, sets for managing unique elements, and sorted sets for ordered data. Redis's popularity is partly due to its support for these complex data structures, enabling developers to employ it for a wide array of applications beyond simple key-value storage.

A notable feature of Redis is its support for atomic operations, crucial for maintaining data integrity, especially in high-concurrency environments. These operations ensure that tasks like incrementing a key's value or adding an element to a list are isolated from other operations. Redis is commonly used as a caching solution, storing frequently accessed data in volatile memory to drastically reduce data retrieval times compared to database querying. This caching mechanism is instrumental in enhancing web application performance by significantly lowering response times and reducing backend database load. Besides serving as a database and cache, Redis functions as a message broker, supporting various messaging patterns like publish/subscribe. This system is vital for developing scalable, distributed applications and implementing event-driven architectures. Redis excels in scalability and high availability, offering features like Redis Sentinel for high availability and Redis Cluster for automatic partitioning. These features enable fault-tolerant, scalable clustered configurations to handle growing data and traffic. Redis's simplicity and ease of use, combined with its minimalistic design and simple command set, make it accessible for integration into various applications. Its single-threaded event-loop architecture is particularly effective in scenarios that demand rapid data access, handling millions of requests per second with minimal latency.

Security features in Redis, including client-side authentication and SSL encryption support, ensure secure connections, vital for applications dealing with sensitive data. Redis's flexibility is further demonstrated by its support for multiple programming languages, making it easily integrable into diverse application stacks. In real-world applications, Redis is utilized across various domains, including e-commerce platforms for session caching and real-time analytics, gaming applications for leaderboards and session storage, and financial services for high-speed transactions and fraud detection. Its versatility in handling diverse data structures and performing rapid operations makes it a powerful tool for addressing complex software development challenges.

3.1.8 Gunicorn

Gunicorn[21], short for 'Green Unicorn,' is a prominent Python Web Server Gateway Interface (WSGI) HTTP server known for its robustness, efficiency, and high configurability. It is particularly favored for deploying Python web applications. Gunicorn serves as a server that translates HTTP requests into Python calls in accordance with the WSGI specification, making it compatible with numerous Python web frameworks such as Django, Flask, and Pyramid. The key strength of Gunicorn lies in its simplicity and ease of use. It is designed for straightforward implementation with sensible default configurations suitable for most Python web applications. However, Gunicorn also caters to more complex requirements through extensive customization options, allowing fine-tuning of parameters like worker processes, worker class, timeout settings, and logging. This balance of simplicity and configurability positions Gunicorn as an apt choice for both basic and intricate deployments. Gunicorn operates on a pre-fork worker model, forking several worker processes that handle requests. This model is particularly efficient for CPU-bound and I/O-bound applications, with the parallel running workers enhancing the server's capacity to manage multiple requests simultaneously, making it well-suited for high concurrency environments. Notably, Gunicorn supports different types of worker classes. While synchronous workers are the default, it also offers support for asynchronous workers through gevent or eventlet for applications requiring real-time interactions like long-polling or WebSockets.

In terms of performance, Gunicorn provides a balance between speed, resource efficiency, and usability. It can handle a high volume of requests with relatively low latency, an essential attribute for modern web applications. Often, Gunicorn is used alongside other web servers like Nginx or Apache, where it complements their strengths in managing static content and connections, with its own ability to serve dynamic Python applications. Security within Gunicorn is a considered aspect, especially since it's expected to run behind a reverse proxy in production. It supports secure HTTP headers and SSL for encrypted connections between the reverse proxy and Gunicorn, ensuring the security of data in transit. For monitoring and management, Gunicorn offers detailed error logs and integration with application performance monitoring tools, crucial for diagnosing issues, understanding application performance, and ensuring smooth operations in production environments. Gunicorn's design philosophy, which emphasizes simplicity and pragmatism, aligns well with the Python community's values of readability and simplicity, making it an appealing tool for developers and system administrators.

In practical applications, Gunicorn is widely utilized in small to medium-sized deployments and can be scaled for larger applications with appropriate configuration and resource allocation. Its compatibility with various web frameworks and ease of integration into Python projects make it a preferred choice for Python developers seeking a reliable and efficient WSGI server.

3.1.9 Docker-Compose

Docker-Compose[22] is a pivotal tool within the Docker ecosystem, streamlining the management of multi-container Docker applications. Its role is especially significant in complex application environments, where it simplifies and optimizes container orchestration. At the heart of Docker-Compose is the ability to define a multi-container application in a single file, typically named `docker-compose.yml`. This YAML-formatted file outlines the application's infrastructure, including services, networks, and volumes. This arrangement provides a straightforward and clear approach to deploying applications, with the ability to start up the entire application using just the `docker-compose up` command. A key feature of Docker-Compose is its service definition capabilities. Within the `docker-compose.yml` file, each service corresponds to a container that runs a specific part of the application, like a web server, database, or caching server. Docker-Compose enables the definition of these services, their configurations, interdependencies, and the sequence of their initiation. This orchestration is essential for ensuring coordinated launching and scaling of application components[23].

Networking between containers is efficiently handled by Docker-Compose, which sets up a unified network for the application. Containers join this network, allowing inter-container communication through service names. This simplification is crucial for most multi-container applications, facilitating container linking and communication. Volume management is another significant utility of Docker-Compose. It allows the definition and mounting of volumes to ensure data persistence, even if containers are destroyed. This feature is especially vital for stateful applications, such as databases, where data persistence is key. Docker-Compose also excels in environment management. It enables the specification of environment variables directly in the `docker-compose.yml` file or in external files, ensuring consistent environmental settings across development, testing, and production stages. This consistency reduces the chances of encountering the “it works on my machine” syndrome.

In terms of scalability, Docker-Compose offers features that allow for the easy scaling of services up or down with a single command. This scalability is particularly beneficial

in development and testing environments, where configurations may need to be rapidly adjusted. Seamless integration with Docker Swarm is another advantage of Docker-Compose, facilitating a smooth transition from single-node setups to multi-node setups in production environments. For debugging and logging, Docker-Compose provides comprehensive logging features that aggregate logs from all containers, offering a centralized view for monitoring and troubleshooting. The strength of Docker-Compose also lies in its robust community and ecosystem. As an open-source project, it benefits from a vast user and contributor base. The community-generated content, including various docker-compose files for common application stacks, provides a rich resource for developers. In practical applications, Docker-Compose is extensively used in microservices architectures and is a popular choice in development environments for running multi-container applications on local machines.

3.1.10 Prometheus

Prometheus[24], an open-source systems monitoring and alerting toolkit, has become a key player in IT infrastructure management, known for its robustness, scalability, and adept handling of time-series data[25]. At its core, Prometheus excels in collecting and storing metrics as time-series data, comprising timestamped values and key-value pair labels. It employs a pull model over HTTP, scraping metrics from services at defined intervals, a method that stands apart from traditional push-based monitoring systems and is particularly effective in dynamic environments, such as those using container orchestration platforms like Kubernetes. Prometheus's data model and query language (PromQL) are central to its functionality. PromQL facilitates complex queries for real-time monitoring, alerting, and historical analysis, making Prometheus versatile for various use cases, from simple metric aggregation to intricate monitoring scenarios involving multiple data sources.

A critical aspect of Prometheus is its alerting functionality. It evaluates rules on time-series data to identify issues, triggering alerts managed by Alertmanager, a key component of the Prometheus ecosystem. Alertmanager oversees alert routing, grouping, and silencing, and integrates with various notification channels to ensure efficient and timely alerts. Prometheus is designed for scalability, handling high-dimensional data from thousands of services and systems without sacrificing performance. This scalability makes it suitable for both small and large-scale monitoring needs. Its decentralized architecture allows for the deployment of multiple independent Prometheus servers, each monitoring specific targets, enhancing scalability and fault tolerance. Integration capabilities are another strong suit of Prometheus. It seamlessly integrates with a wide array of services and systems,

including various service discovery mechanisms. In Kubernetes environments, Prometheus can automatically discover and monitor new pods or services, making it well-suited for modern, dynamic environments.

Visualization is a crucial component of monitoring, and Prometheus integrates effortlessly with tools like Grafana. This integration enables the creation of detailed dashboards that offer visual insights into collected metrics, aiding in the identification of trends, patterns, and potential issues. Reliability is also a key design element of Prometheus. Its local storage is fault-tolerant, handling crashes and disk failures. While it doesn't provide a clustered storage solution natively, Prometheus supports integration with remote storage options for long-term storage and enhanced availability. The Prometheus community is a significant strength, contributing actively to its development, documentation, and support. This vibrant community ensures the continuous evolution of Prometheus, with regular additions of new features and improvements. In practical applications, Prometheus is widely used across various industries for diverse monitoring purposes. It is particularly popular in cloud-native environments due to its service discovery and dynamic monitoring capabilities. Prometheus is also extensively used in traditional server environments, application monitoring, and IoT scenarios, showcasing its adaptability and versatility.

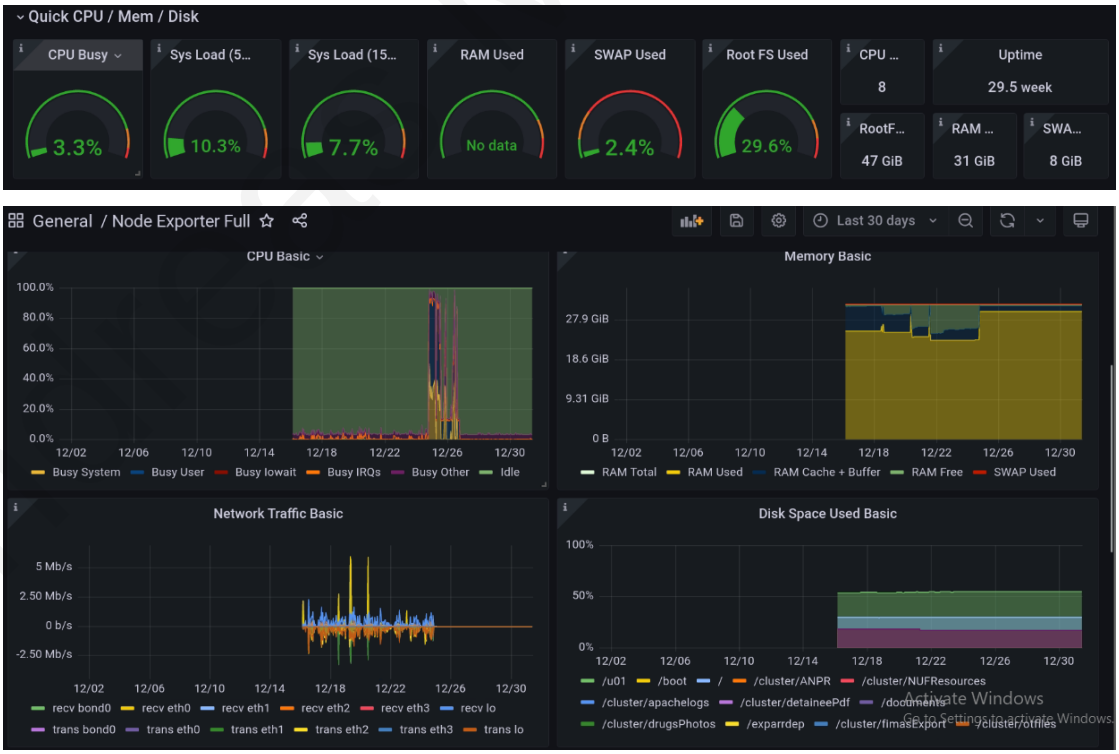


Figure 7 Prometheus - Grafana Dashboard

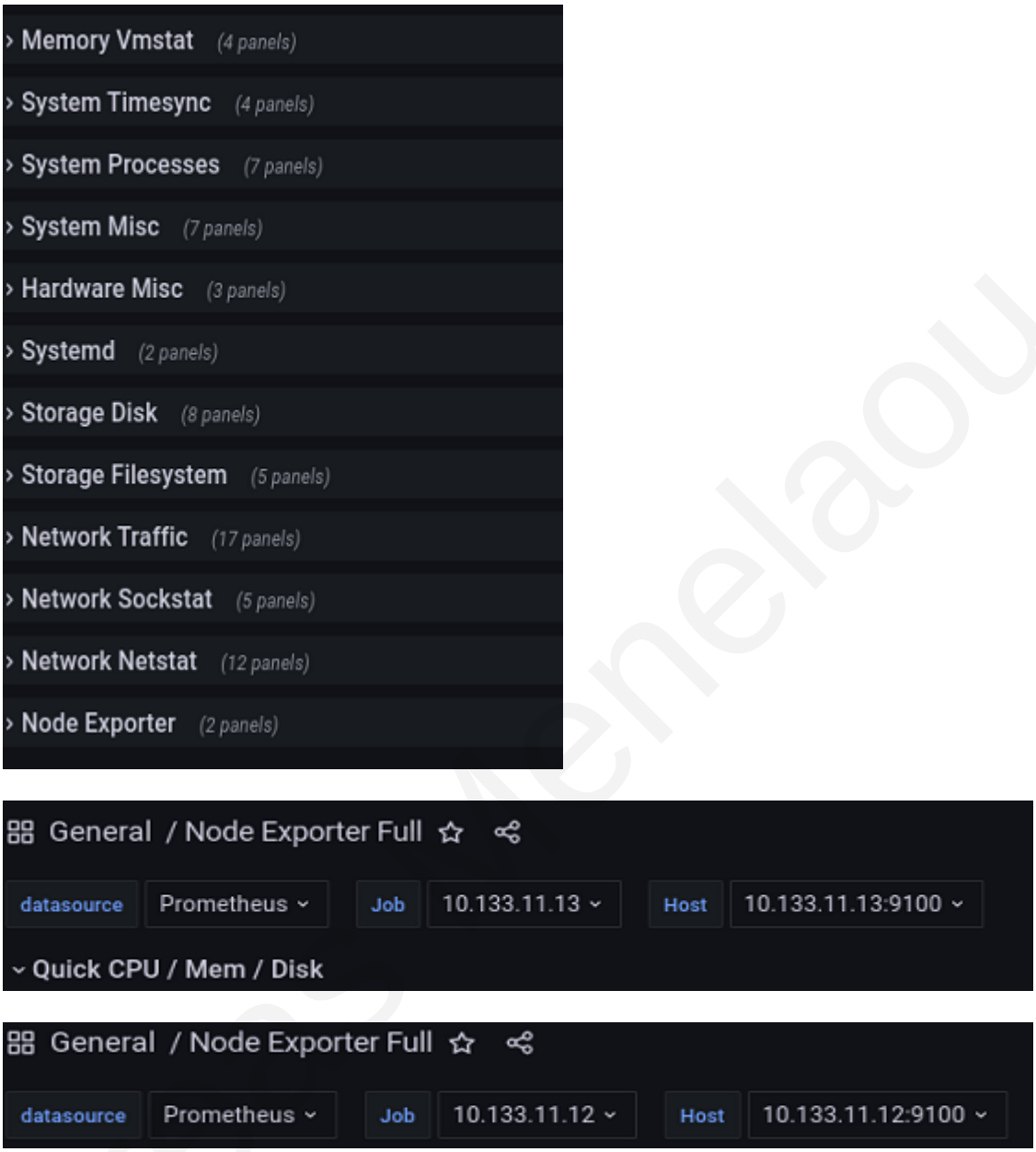


Figure 8 Prometheus servers- Grafana Dashboard

3.1.11 Node Exporter

Node Exporter[26], a vital part of the Prometheus monitoring ecosystem, is designed to collect and display an extensive range of hardware and operating system metrics on Unix/Linux systems. As a server, Node Exporter is lightweight yet powerful, efficiently gathering system-level metrics and exposing them via HTTP for Prometheus to scrape. Its implementation in Go ensures both efficiency and ease of deployment, typically as a single static binary. The primary function of Node Exporter is to gather a broad spectrum of system metrics, including CPU usage (divided by mode), memory utilization, disk space, I/O statistics, network bandwidth, and error statistics. This

comprehensive coverage is invaluable for system administrators and developers, providing deep insights into system performance and health.

Node Exporter's simplicity and ease of use are key aspects of its design. It requires minimal configuration and can be deployed quickly, making it an ideal tool for a variety of environments, including those undergoing rapid scaling or experiencing dynamic infrastructure changes. Once operational, it starts collecting metrics automatically, streamlining the setup process and enhancing usability in complex environments. A significant strength of Node Exporter lies in its ability to present detailed system metrics in a format easily processed by Prometheus. This capability allows Prometheus to aggregate, store, and analyze the data, enabling the creation of comprehensive monitoring dashboards, alert setups, and long-term trend analysis. Together, Node Exporter and Prometheus form a robust platform for monitoring server health and performance. In terms of scalability, Node Exporter is designed to be both lightweight and efficient, suitable for running on a wide range of systems from small single-board computers to large servers without significantly impacting performance. This scalability ensures its applicability across both small-scale applications and large, distributed systems.

Node Exporter also provides extensibility through various collectors, which can be enabled or disabled to tailor the metrics to specific needs. This feature allows for customization in monitoring, from basic system metrics to detailed performance analysis, including specific hardware monitoring like temperature sensors or detailed network metrics. Reliability is another critical aspect of Node Exporter, designed to run as a daemon and consistently collect and expose metrics. This continuous operation is crucial for effective monitoring, ensuring consistent data availability for Prometheus. Seamless integration with Prometheus is achieved through service discovery or static configuration, enabling Prometheus to adapt to infrastructure changes, such as adding or removing servers. The vibrant community and ecosystem surrounding Node Exporter, being open-source, benefit from contributions that continually enhance its capabilities, introduce new features, and offer support. This active community participation ensures Node Exporter remains current with technological trends and monitoring needs.

Figure 9 Node Exporter

3.1.12 Alertmanager

Alertmanager[27], a crucial component of the Prometheus monitoring toolkit, is tailored to manage alerts generated by client applications like the Prometheus server. Its function in sophisticated alert handling mechanisms is vital for maintaining high availability and performance in systems. Alertmanager enhances Prometheus's monitoring capabilities by specializing in the processing of generated alerts, efficiently handling data collection and alert processing separately. One of the defining features of Alertmanager is its ability to deduplicate alerts. In dynamic, distributed environments, it's common for multiple Prometheus instances to send similar alerts regarding the same issue. Alertmanager effectively deduplicates these alerts, ensuring that operators receive a single notification for each issue, thus reducing noise and preventing alert fatigue. Additionally, Alertmanager groups similar alerts, consolidating them into a single notification based on customizable criteria. This grouping makes alerts more manageable and helps in understanding the overall context of a problem.

Alert routing in Alertmanager is highly flexible, allowing the routing of different types of alerts to various receivers based on severity, affected systems, or other defined criteria. This flexibility ensures that critical alerts are sent to the appropriate personnel or systems, enabling quick and effective responses. For example, critical alerts can be directed to an on-call engineer via PagerDuty, while less critical alerts might be sent to

a Slack channel or an email list. Alertmanager also excels in alert management during maintenance periods or in the presence of known issues. Users can silence alerts based on specific criteria, avoiding notifications for non-actionable alerts. The inhibition feature suppresses notifications for certain alerts when more critical alerts are firing, preventing an overflow of notifications due to a primary issue causing secondary problems. Integration with various notification channels is a significant aspect of Alertmanager's functionality. It supports a wide range of integrations, including email, Slack, PagerDuty, OpsGenie, and more. This extensive support enables Alertmanager to fit into different organizational workflows and communication practices, accommodating teams of various sizes and structures.

In terms of reliability, Alertmanager is designed for high availability. It supports running multiple instances in a clustered setup, ensuring resilience and continuous alert processing, even if one instance fails. The configurability of Alertmanager is both flexible and powerful, allowing the definition of complex routing rules, grouping parameters, and receiver configurations to tailor Alertmanager to the specific needs and workflows of different organizations.

3.1.13 Django

Django[28], a high-level Python web framework, is highly esteemed for its facilitation of rapid development and clean, pragmatic design. As a free and open-source tool, it simplifies the creation of complex, data-driven websites. Django's functionality is anchored in its Model-View-Template (MVT) architecture, a variation of the Model-View-Controller (MVC) architecture. This architecture distinctly separates data handling (Model), user interface (Template), and business logic (View), fostering a modular and scalable design that is conducive to changes and maintenance. The Model layer abstracts the database schema, allowing interaction without direct SQL queries, while the View layer manages business logic and HTTP interactions. The Template layer, meanwhile, handles the generation of user-facing elements.

The framework's Object-Relational Mapping (ORM) is another acclaimed feature, offering a powerful database abstraction API for creating, retrieving, updating, and deleting objects. This ORM significantly reduces the need for boilerplate database code, allowing for more Pythonic interaction with the database. Django's "batteries-included" philosophy means it comes equipped with a suite of built-in features for common web development needs, including an authentication system, URL routing, a template engine, ORM, and database schema migrations. These integrated components streamline the development process. Furthermore, Django's admin

interface, known for its automatic generation and customizability, provides a user-friendly interface for site content management. Security is a paramount concern in Django, which offers built-in protections against various threats such as XSS, CSRF, SQL Injection, and Clickjacking, alongside a secure system for managing user accounts and passwords.

In terms of scalability, Django excels at handling high traffic and large-scale applications. It can be scaled up or out to meet increasing demands, supporting enterprise-level applications and sites with heavy traffic. The framework is also highly extensible and customizable, thanks to its support for class-based views and an array of middleware, allowing developers to enhance or add new functionalities as needed. Django promotes rapid development due to its many out-of-the-box solutions and clean, model-based approach, significantly shortening development cycles. Its vibrant community and comprehensive documentation, stemming from its open-source nature, provide invaluable support and contribute to its continual evolution with the latest web development trends.

The Django REST framework, essential for building Web APIs, especially for Single Page Applications (SPAs) and mobile apps, offers flexibility and a set of tools for creating web-browsable APIs, greatly aiding in API testing and debugging.

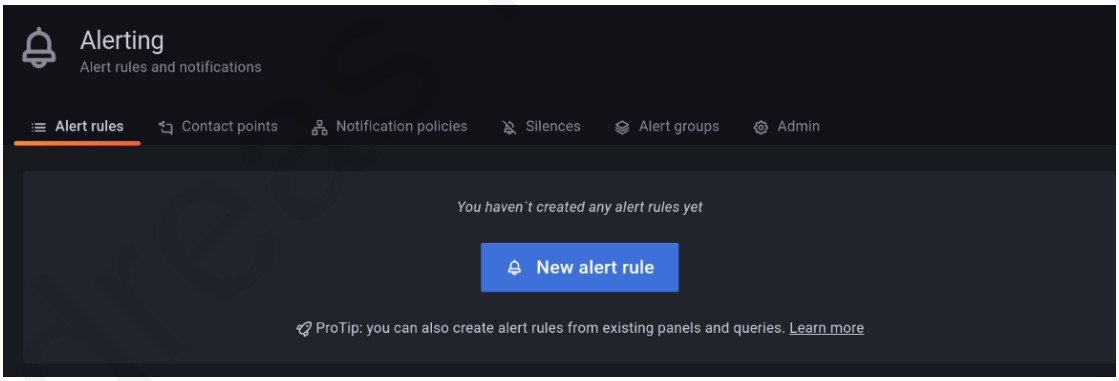


Figure 10 Prometheus alerting setup- Grafana Dashboard

Alertrules file

Chapter 4 - Analysis

The Prometheus client libraries offer four core metric types. These are currently only differentiated in the client libraries (to enable APIs tailored to the usage of the specific types) and in the wire protocol.

Use cases for counters include request count, tasks completed, and error count.

4.1 Counters

Have to be processed – No repetition and no statistical value in that. Use e.g derivative function to convert a counter to gauge equivalent

This cumulative metric is suitable for tracking the number of requests, errors or completed tasks. It cannot decrease, and must either go up or be reset to zero.

Counters should be used for:

- Recording a value that only increases
- Assessing the rate of increase (later queries can show how fast the value rises)

4.2 Gauge

This point-in-time metric can go both up and down. It is suitable for measuring current memory use and concurrent requests.

Gauges should be used for:

- Recording a value that may go up or down
- Cases where you don't need to query the rate of the value

Use cases for gauges include queue size, memory usage, and the number of requests in progress.

4.3 Histogram

This metric is suitable for aggregated measures, including request durations, response sizes that measure application performance. Histograms sample observations and categorize data into buckets that you can customize.

Histograms should be used for:

- Multiple measurements of a single value, allowing for the calculation of averages or percentiles
- Values that can be approximate
- A range of values that you determine in advance, by using default definitions in a histogram bucket, or your custom values

Use cases for histograms include request duration and response size.

4.4 Summary

This metric is suitable for accurate quartiles. A summary samples observations and provides a total count of observations, as well as a sum of observed values, and calculates quartiles.

Summaries should be used for:Multiple measurements of a single value, allowing for the calculation of averages or percentiles Values that can be approximate A range of values that you cannot determine upfront, so histograms are not appropriate

Use cases for summaries include request duration and response size.

4.5 Common Prometheus Use Cases and Associated Metrics

Here are a few common use cases of Prometheus, and the metrics most appropriate to use in each case.

CPU Usage

The metric used here is “node_cpu_seconds_total”. This is a counter metric that counts the number of seconds the CPU has been running in a particular mode. The CPU has several modes such as iowait, idle, user, and system. Because the objective is to count usage, use a query that excludes idle time:

```
sum by (cpu)(node_cpu_seconds_total{mode!="idle"})
```

The sum function is used to combine all CPU modes. The result shows how many seconds the CPU has run from the start. To tell if the CPU has been busy or idle recently, use the rate function to calculate the growth rate of the counter:

```
(sum by (cpu)(rate(node_cpu_seconds_total{mode!="idle"}[5m]))*100
```

The above query produces the rate of increase over the last five minutes, which lets you see how much computing power the CPU is using. To get the result as a percentage, multiply the query by 100.

Memory Usage

The following query calculates the total percentage of used memory:

```
node_memory_Active_bytes/node_memory_MemTotal_bytes*100
```

To obtain the percentage of memory use, divide used memory by the sum and multiply by 100.

Free Disk

You need to know your free disk usage to understand when there needs to be more space on the infrastructure nodes. Again, the same memory usage method is used here, but with different metric names.

```
node_filesystem_avail_bytes/node_filesystem_size_bytes*100
```

4.6 Deep Analysis

What is already available.

. We have collected from the exporter data. Numerical data.

. We can apply mathematical functions on it and create alerts and rule model

. Chart Histogram with Grafana.

What data to export?

. Raw metrics data, no functions applied on it

. As much as possible

Two ways to get data out of promertheous

. HTTP API(Poll) – Exploratory data analysis

```
requests.get(
  url = 'http://127.0.0.1:9090/api/v1/query_range',
  params = {
    'query': 'sum({__name__=~".+"})  by (__name__,instance
    'start': '1502809554',
    'end'  : '1502839554',
    'step' : '1m'
  })

{"data": {...,  "resultType": "matrix",
"result": [{
  "metric": {"method": "GET",...},
  "values": [[1500008340,"3"], ... ]},...]
}}
```

. Remote API (Push) – Streaming analysis

Easiest way to export is Grafana and Python (robust perception blog entry)

CSV file is large so to reduce data I use domain knowledge to select relevant data subset

{__name__=~".+"}

Use alerts as initial set of training labels

File: hostMonitoring.rules

Y= ALERTS{name="high_latency"} tidy up, verify true positives, annotate manually to create training and matrix data

File: ***hostMonitoring.rules***

File: ***ServerData.csv and andreas.py***

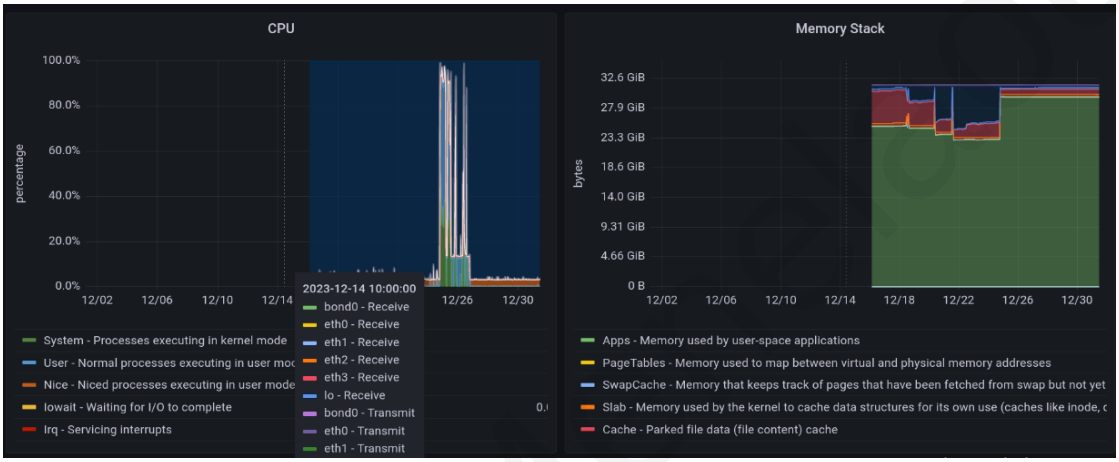


Figure 11 Prometheus CPU & Memory Stack- Grafana Dashboard

Figure 12 Prometheus Network Traffic & Disk Space Used- Grafana Dashboard

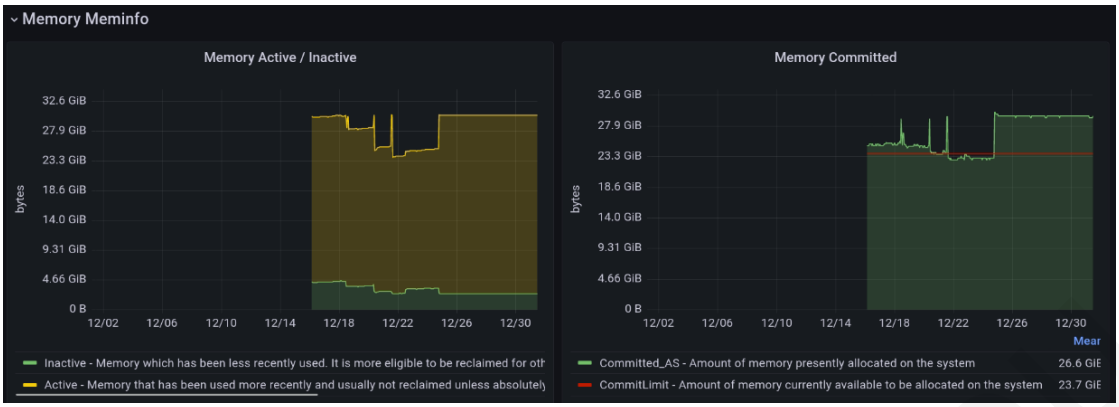


Figure 13 Prometheus Memory Active & Committed- Grafana Dashboard

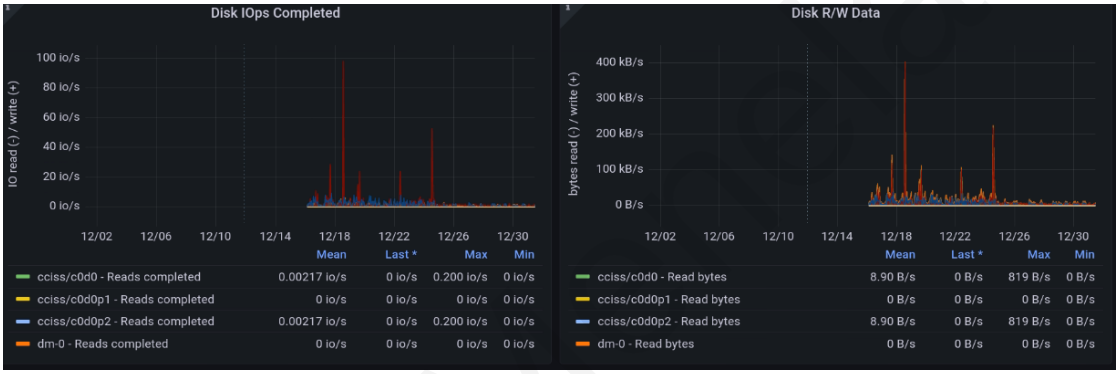


Figure 14 Disk IOps Completed & Disk R/W Data - Grafana Dashboard

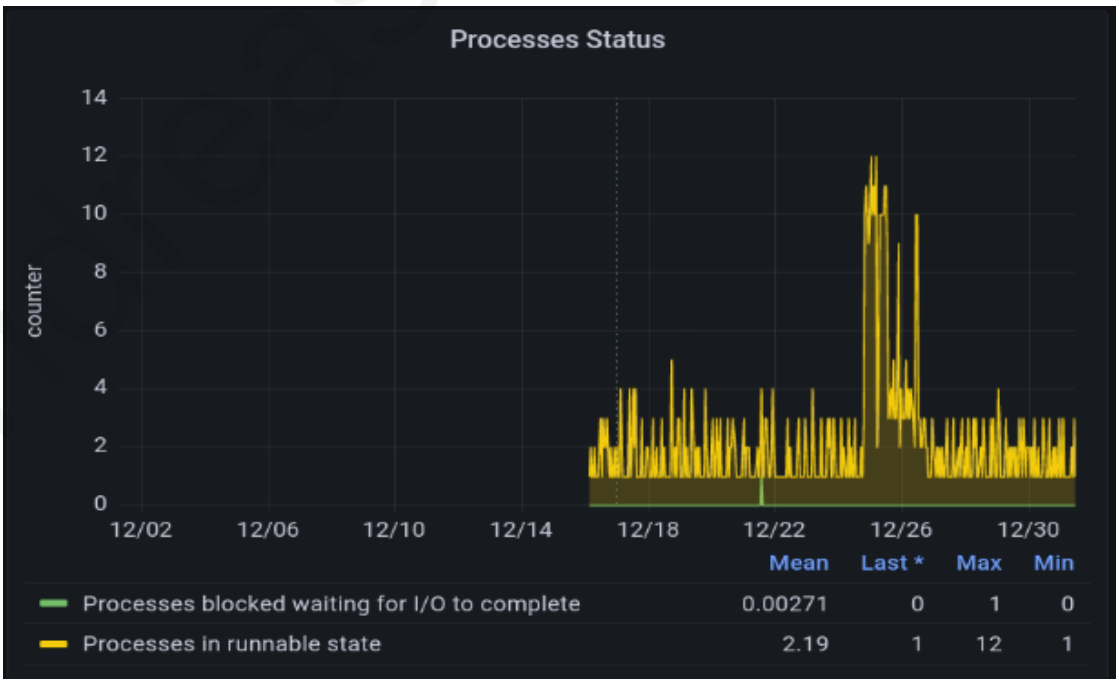
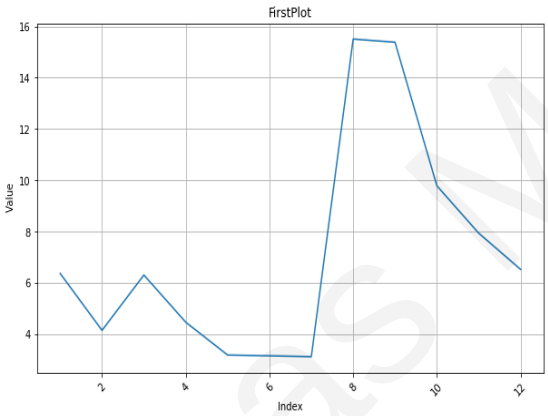


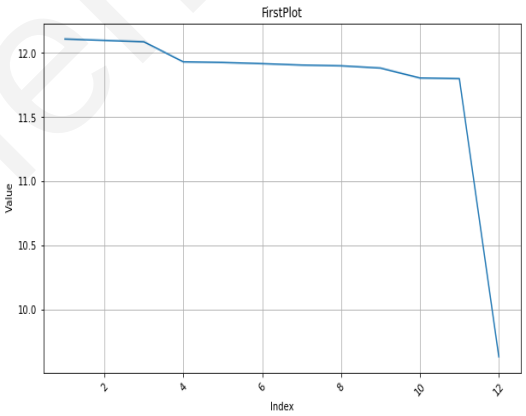
Figure 15 Prometheus Processes Status- Grafana Dashboard

```
groups:
- name: Applications Servers
  rules:
  - alert: Host Out Of Memory
    expr: node_memory_MemFree_bytes / node_memory_MemTotal_bytes * 100 < 10
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: Application Server1 out of memory
      description: "Node memory is filling up. If for 5 minutes the percentage of memory left is less than 10%"
```

Application Server1



Application Server 2



groups:

- name: Applications Servers

- alert: Host Out Of Disk Space

  expr: (node_filesystem_avail_bytes * 100) / node_filesystem_size_bytes < 10

  for: 2m

  labels:

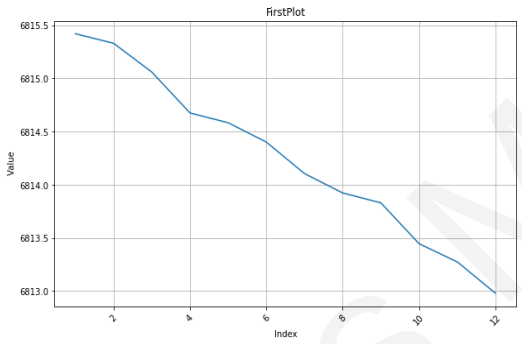
    severity: warning

  annotations:

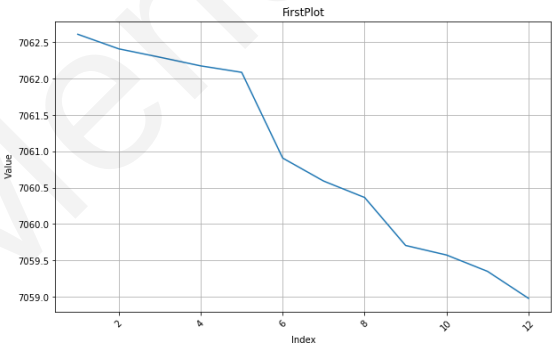
    summary: Host out of disk space

    description "Disk is almost full"

Application Server1



Application Server 2



groups:

- name: Applications Servers

alert: HostSwapIsFillingUp

expr: (1 - (node_memory_SwapFree_bytes / node_memory_SwapTotal_bytes)) * 100 > 80

for: 2m

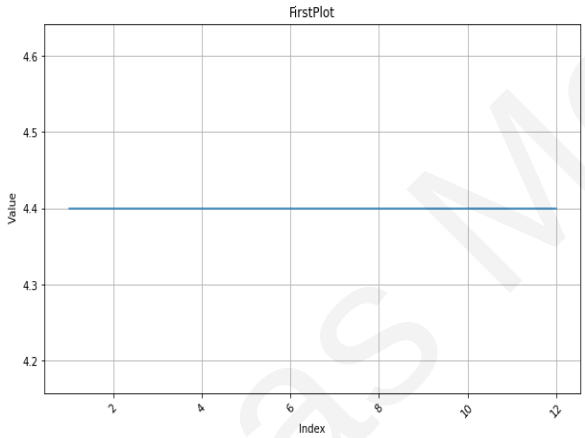
labels:

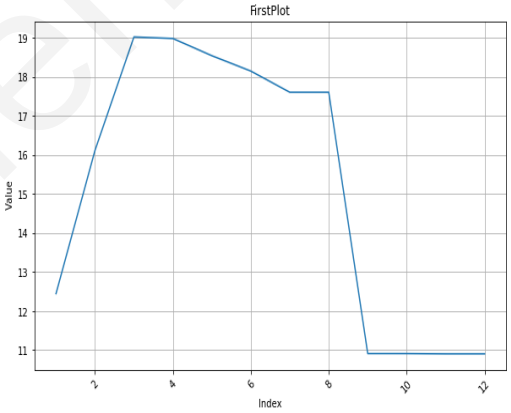
severity: warning

annotations:

summary: Host swap is filling up

description: "Swap is filling up (>80%)"





Conclusion

Deploying the Ticket Desk application into the Police IT environment involves a systematic process of preparation, configuration, deployment, testing, and integration. Successful deployment requires meticulous attention to detail, ensuring compatibility, proper configurations, and thorough testing to guarantee a seamless transition into the production environment. Post-deployment, it's crucial to maintain documentation, provide training, and establish a support system for ongoing maintenance. By following these steps diligently, the Kios Ticket Desk application can be seamlessly integrated into the IT environment, enhancing operational efficiency and support capabilities.

Considering future improvements or integration tasks after the initial deployment is crucial. This involves adding features or modules for improved functionality. Furthermore, scaling and performance Optimization scalability as infrastructure demand increases. This will optimize performance by fine-tuning application components and infrastructure. Conduct regular security audits and updates to safeguard against potential threats and implement additional security measures as per evolving industry standards. Gather ICT operators feedback to enhance the application ‘s usability and interface. Implement UX enhancements to streamline user interactions, testing processes and efficient updates. Another important feature aspect is Analytics and reporting. By integrating analytics tools to gather insights and generate reports for informed decision-making. Future work also includes ongoing training sessions and support to users for maximizing the application’s potential. Stay updated with industry regulations and ensure the application complies with evolving standards. Finally implement robust backup systems and disaster recovery plans to mitigate data loss risks.

References

[1] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, Jun. 2013, doi: 10.1016/J.COMNET.2013.04.001.

[2] A. Iyengar, E. MacNair, and T. Nguyen, "Analysis of web server performance," *Conference Record / IEEE Global Telecommunications Conference*, vol. 3, pp. 1943–1947, 1997, doi: 10.1109/GLOCOM.1997.644616.

[3] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "CPU demand for web serving: Measurement analysis and dynamic estimation," *Performance Evaluation*, vol. 65, no. 6–7, pp. 531–553, Jun. 2008, doi: 10.1016/J.PEVA.2007.12.001.

[4] H. Liu, "A measurement study of server utilization in public clouds," *Proceedings - IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, DASC 2011*, pp. 435–442, 2011, doi: 10.1109/DASC.2011.87.

[5] "Infrastructure Monitoring Challenges and How to Tackle Them - VirtualMetric - Infrastructure Monitoring Blog." Accessed: Dec. 20, 2023. [Online]. Available: https://www.virtualmetric.com/blog/infrastructure-monitoring-challenges

[6] "Server Monitoring: Benefits and Challenges - Alvaka." Accessed: Dec. 20, 2023. [Online]. Available: https://www.alvaka.net/server-monitoring-benefits-and-challenges/

[7] "Server Monitoring: Top Five Issues | FrameFlow." Accessed: Dec. 20, 2023. [Online]. Available: https://www.frameflow.com/blog/top-five-issues-detected-by-server-and-it-systems-monitoring-software/

[8] "Docker: Accelerated Container Application Development." Accessed: Dec. 20, 2023. [Online]. Available: https://www.docker.com/

[9] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, Jan. 2015, doi: 10.1145/2723872.2723882.

[10] "(PDF) An Introduction to Docker and Analysis of its Performance." Accessed: Dec. 20, 2023. [Online]. Available: https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance

[11] T. Bui, "Analysis of Docker Security," Jan. 2015, Accessed: Dec. 20, 2023. [Online]. Available: https://arxiv.org/abs/1501.02967v1

[12] “HAProxy - The Reliable, High Perf. TCP/HTTP Load Balancer.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.haproxy.org/

[13] J. E. C. De La Cruz and I. C. A. R. Goyzueta, “Design of a high availability system with HAProxy and domain name service for web services,” *Proceedings of the 2017 IEEE 24th International Congress on Electronics, Electrical Engineering and Computing, INTERCON 2017*, Oct. 2017, doi: 10.1109/INTERCON.2017.8079712.

[14] “PostgreSQL: The world’s most advanced open source database.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.postgresql.org/

[15] M. Stonebraker and L. A. Rowe, “The design of POSTGRES,” *ACM SIGMOD Record*, vol. 15, no. 2, pp. 340–355, Jun. 1986, doi: 10.1145/16856.16888.

[16] “Nginx: the High-Performance Web Server and Reverse Proxy.” Accessed: Dec. 20, 2023. [Online]. Available: https://dl.acm.org/doi/fullHtml/10.5555/1412202.1412204

[17] “Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.nginx.com/

[18] “Celery - Distributed Task Queue — Celery 5.3.6 documentation.” Accessed: Dec. 20, 2023. [Online]. Available: https://docs.celeryq.dev/en/stable/

[19] “Periodic Tasks — Celery 5.3.6 documentation.” Accessed: Dec. 20, 2023. [Online]. Available: https://docs.celeryq.dev/en/stable/userguide/periodic-tasks.html

[20] “Redis.” Accessed: Dec. 20, 2023. [Online]. Available: https://redis.io/

[21] “Gunicorn - Python WSGI HTTP Server for UNIX.” Accessed: Dec. 20, 2023. [Online]. Available: https://gunicorn.org/

[22] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, “Developing Docker and Docker-Compose Specifications: A Developers’ Survey,” *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2021.3137671.

[23] M. H. Ibrahim, M. Sayagh, and A. E. Hassan, “A study of how Docker Compose is used to compose multi-component systems,” *Empir Softw Eng*, vol. 26, no. 6, pp. 1–27, Nov. 2021, doi: 10.1007/S10664-021-10025-1/FIGURES/19.

[24] “Prometheus - Monitoring system & time series database.” Accessed: Dec. 20, 2023. [Online]. Available: https://prometheus.io/

[25] B. Rabenstein and J. Volz, “Prometheus: A {Next-Generation} Monitoring System (Talk).” 2015.

[26] “prometheus/node_exporter: Exporter for machine metrics.” Accessed: Dec. 20, 2023. [Online]. Available: https://github.com/prometheus/node_exporter

[27] “Alertmanager | Prometheus.” Accessed: Dec. 20, 2023. [Online]. Available: https://prometheus.io/docs/alerting/latest/alertmanager/

[28] “The web framework for perfectionists with deadlines | Django.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.djangoproject.com/

Appendix A – Server Installation Manual

Step 1: Network Check

Check if network is connected by running the command:

```
nmcli device status
```

If the network is not connected then run the command:

```
nmtui
```

Check the connection settings and perform the appropriate changes, exit nmtui and run:

```
systemctl restart NetworkManager
```

Step 2: Update Rocky Linux

Run the command:

```
sudo yum update
```

Step 3: Installation of Docker CE

Run the following commands:

```
sudo dnf config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo
sudo dnf makecache
sudo dnf install -y docker-ce
sudo systemctl start docker.service
sudo systemctl status docker.service
sudo systemctl enable --now docker.service
sudo docker version
```

Step 4: Test Docker CE Installation

Run the following commands:

```
create docker to test
docker search alpine --filter is-official=true
docker pull alpine
docker images
docker run -it --rm alpine /bin/sh
```

Step 5: Docker Compose Installation

Run the following commands:

```
sudo curl -L https://github.com/docker/compose/releases/download/v2.15.1/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

docker-compose version
```

Step 6: PostgreSQL 14 Installation Using RPM

Run the following commands:

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

sudo dnf -qy module disable postgresql

sudo dnf install -y postgresql14-server

sudo /usr/pgsql-14/bin/postgresql-14-setup initdb

sudo systemctl start postgresql-14

sudo systemctl status postgresql-14

sudo systemctl enable postgresql-14
```

Step 7: Test PostgreSQL Installation

Run the following commands to check if PostgreSQL has been installed correctly:

```
sudo -i -u postgres

psql

\q
```

Step 8: Create Ticketing Database

Run the following commands: (make sure to change <USERNAME> and <PASSWORD> with the relevant credentials)

```
sudo -i -u postgres

psql

CREATE DATABASE ticketingdb;

CREATE USER <USERNAME> WITH ENCRYPTED PASSWORD '<PASSWORD>';

GRANT ALL PRIVILEGES ON DATABASE ticketingdb TO <USERNAME>;
```

Step 9: Edit PostgreSQL Config

Run the following command and add the following text in the configuration file pg_hba.conf

```
sudo -i -u postgres
nano /var/lib/pgsql/14/data/pg_hba.conf
```

Add the following lines

```
host all all 127.0.0.1/32 md5
host ticketingdb <USERNAME> 192.168.250.0/25 trust
```

Save and exit and restart PostgreSQL

Step 10: HAProxy Installation

Run the following commands

```
sudo dnf install gcc pcre-devel tar make openssl-devel
readline-devel systemd-devel wget vim
wget https://www.lua.org/ftp/lua-5.3.5.tar.gz
tar xzf lua-5.3.5.tar.gz
cd lua-5.3.5 && sudo make linux install
sudo useradd -M -d /var/lib/haproxy -s /sbin/nologin -r haproxy
cd ..
wget http://www.haproxy.org/download/2.6/src/haproxy-
2.6.5.tar.gz
tar xzf haproxy-2.6.5.tar.gz
cd haproxy-2.6.5
make -j $(nproc) TARGET=linux-glibc USE_OPENSSL=1 USE_LUA=1
USE_PCRE=1 USE_SYSTEMD=1 USE_PROMEX=1
sudo make install
mkdir /etc/haproxy/
cd /etc/haproxy/
vi haproxy.cfg
```

Then allow HAProxy through the firewall using the following commands:

```
sudo firewall-cmd --add-port=8404/tcp --permanent
sudo firewall-cmd --reload
mkdir /var/lib/haproxy/
haproxy -c -f /etc/haproxy/haproxy.cfg
sudo firewall-cmd --add-port=80/tcp --permanent
```

```
sudo firewall-cmd --add-port=443/tcp --permanent
sudo firewall-cmd –reload
```

Create HAProxy systemd Configuration file

```
sudo nano /etc/systemd/system/haproxy.service
```

Write the following content

```
[Unit]
Description=HAProxy Load Balancer
After=network-online.target
Wants=network-online.target

[Service]
Environment="CONFIG=/etc/haproxy/haproxy.cfg"
"PIDFILE=/run/haproxy.pid"
EnvironmentFile=/etc/sysconfig/haproxy
ExecStartPre=/usr/local/sbin/haproxy -f $CONFIG -c -q $OPTIONS
ExecStart=/usr/local/sbin/haproxy -Ws -f $CONFIG -p $PIDFILE
$OPTIONS
ExecReload=/usr/local/sbin/haproxy -f $CONFIG -c -q $OPTIONS
ExecReload=/bin/kill -USR2 $MAINPID
SuccessExitStatus=143
KillMode=mixed
Type=notify

[Install]
WantedBy=multi-user.target
```

Save and close the file and run the following commands. (Please note that HAProxy will fail to start at command “sudo systemctl start haproxy” due to not having a correct haproxy.cfg file. The haproxy.cfg will be provided with the RESPOnse platform with detailed explanation for each of the configurations enclosed in the file.)

```
echo 'OPTIONS="-Ws"' > /etc/sysconfig/haproxy
sudo systemctl daemon-reload
sudo systemctl start haproxy
```

```
sudo systemctl status haproxy
sudo systemctl enable haproxy
```