



University of Manouba

National School of Computer



Company Immersion Internship Report

High Availability Setup for Keycloak in Kubernetes

Prepared by

MACHRAOUI Mohamed.

Organization



supervised by

MAJDOUB Mahmoud

Academic year 2023/2024

Acknowledgements

I would like to express my deep gratitude to everyone who contributed to the realization of this project. Their support and commitment were essential in successfully completing this work.

I am especially thankful to **Mr. Mahmoud Majdoub**, my internship supervisor, for his unwavering guidance, insightful feedback, and encouragement throughout the entire process. His expertise was invaluable in helping me navigate challenges and achieve my objectives.

I also extend my heartfelt thanks to the entire LinSoft team and everyone who played a role in bringing this project to fruition. Your collaboration and assistance were greatly appreciated.

Table of Content

General Introduction.....	6
1.1 Introduction	7
1.2 Presentation of LinSoft.....	7
1.3 Project Overview and Solution Approach.....	8
1.3.1 Problem Statement	8
1.3.2 Objectives.....	8
1.3.3 State of the Art	9
1.3.4 Our Solution	9
1.4 Tools and Technologies.....	9
1.5 Methodology.....	10
1.5.1 Why SCRUM ?	11
1.5.2 Key Scrum Practices.....	11
1.6 Conclusion	12
2.1 Introduction	13
2.2 DevOps Concept.....	13
2.2.1 DevOps Philosophy.....	13
2.2.2 Containerization	14
2.2.3 Orchestration.....	15
2.2.4 Kubernetes	15
2.2.5 Why Choose Kubernetes?	15
2.2.6 Kubernetes Architecture	16
2.2.7 Kubernetes Objects.....	17
2.2.8 Kubernetes Distributions	18
2.3 What is High Availability?.....	19
2.3.1 Why is High Availability Important?	19
2.3.2 High Availability in Our Project.....	20
2.4 What is Keycloak?	20
2.4.1 What is Identity and Access Management (IAM)?.....	20
2.4.2 IAM benefits	21
2.4.3 Introducing Keycloak.....	21
2.4.4 Overview of Keycloak and Other IAM Solutions.....	22
2.4.5 Why Choose KEYCLOAK ?	23
2.4.6 Installing and running Keycloak	24

2.4.7 Discovering the Keycloak admin and account consoles	25
2.4.7.1 Key Concepts in Keycloak	26
2.5 What is INFINISPAN ?	26
2.5.1 Why Choose Infinispan?	28
2.5.2 Installing and Running Infinispan.....	28
2.6 What is Nextcloud ?	28
2.6.1 Introducing Nextcloud	29
2.6.2 Comparing Nextcloud with Other Solutions	29
2.6.3 Why Choose Nextcloud?	30
2.6.4 Installing and Running Nextcloud	31
2.3 Conception and Architecture of Our Project.....	31
2.3.1 K3s Architecture with High Availability Setup	31
2.3.2 Multi-Node Deployment of Keycloak, Infinispan, and PostgreSQL in Kubernetes.....	33
2.3.3 Nextcloud and Keycloak Integration in a k3s Cluster	35
2.7 Conclusion	37
3.1 Introduction	38
3.2 Work Environment	38
3.3 Infrastructure Setup	39
3.3.1 Configuration of a Load Balancer	39
3.3.2 Configuration of the External Database.....	40
3.3.3 Setting Up the Cluster	41
3.3.3.1 Master Node Configuration	42
3.3.3.2 Worker Node Configuration.....	43
3.3.3.3 Configuration of the Development Machine.....	43
3.4 conclusion	44
4.1 Introduction	45
4.2 Infinispan Deployment	45
4.2.1 Introduction.....	45
4.2.2 Deploying the Infinispan Operator	45
4.3 Deploying Postgres	53
4.3.1 Deploying PostgreSQL in a Kubernetes Environment	53
4.4 Keycloak deployment.....	56
4.4.1Keycloak operator deployment	56
4.3.2 Deploying Keycloak.....	58
4.4.2 Starting keycloak.....	60

General Conclusion.....	73
-------------------------	----

Figure 1 Logo of Linsoft.	8
Figure 2 Scrum development process.....	11
Figure 3 DevOps Workflow Progression.....	14
Figure 4 Docker containers.....	15
Figure 5 Kubernetes architecture	16
Figure 6 Logo of keycloak	21
Figure 7 keycloak login interface	25
Figure 8 logo of infinispan.....	26
Figure 9 logo of nextcloud.....	29
Figure 10 k3s architecture	32
Figure 11 Multi-Node Deployment of Keycloak, Infinispan, and PostgreSQL in Kubernetes	33
Figure 12 Nextcloud and keycloak integration in a k3s cluster	35
Figure 13 K3S architecture.	39
Figure 14 Haproxy configuration.....	40
Figure 15 Configuration of the External Database	41
Figure 16 List of Images	41
Figure 17 Verification of Master Nodes	43
Figure 18 Verification of Worker Nodes.....	43
Figure 19 OVERVIEW OF THE INFINISPAN OPERATOR WORKFLOW IN KUBERNETES....	46
Figure 20 CLI PLUGIN CHECK	47
Figure 21 OPERATOR CHECK PROCESS	48
Figure 22 Infinispan Operator Manifest	49
Figure 23 LIST OF RUNNING PODS	49
Figure 24 KUBERNETES DEPLOYMENTS.....	49
Figure 25 KUBERNETES SERVICES	49
Figure 26 CONFIGMAP MANIFEST FILE.....	50
Figure 27 CONFIGMAP MANIFEST FILE1.....	50
Figure 28 Checking ConfigMap Deployment.....	51
Figure 29 Checking Secrets.....	51
Figure 30 Infinispan Web Interface.....	52
Figure 31 Infinispan Admin Console.....	52
Figure 32 Verifying Cluster Nodes.....	53
Figure 33 PostgreSQL ConfigMap Setup	53
Figure 34 PostgreSQL Deployment Manifest File.....	54
Figure 35 Checking PostgreSQL Pod Status	55
Figure 36 PostgreSQL Persistent Storage Configuration.....	55

Figure 37 PostgreSQL Pvc Configuration	56
Figure 38 CHECKING Pvc POD STATUS	56
Figure 39 CHECKING kubernetes olm.....	57
Figure 40 CHECKING POSTGRESQL services	57
Figure 41 Viewing Keycloak Tables in PostgreSQL	58
Figure 42 Keycloak Secrets	58
Figure 43 Creating Database Secrets for Keycloak.	59
Figure 44 Keycloak Deployment Manifest File	59
Figure 45 Kubernetes StatefulSet Screenshot.	59
Figure 46 Kubernetes pods Screenshot	59
Figure 47 Keycloak Service Manifest.	60
Figure 48 Kubernetes Services Overview.	60
Figure 49 The Keycloak admin console.....	61
Figure 50 : Realm selector.....	62
Figure 51 The Add user form.....	63
Figure 53 Nextcloud PV Manifest File	65
Figure 52 Nextcloud PVC check	65
Figure 54 Nextcloud PVC Manifest File.....	65
Figure 55 Nextcloud PVC status.....	66
Figure 56 Nextcloud deployment Manifest File	66
Figure 57 nextcloud pods status	67
Figure 58 nextcloud welcome page.	67
Figure 59 create realm	68
Figure 60 create client	68
Figure 61 create user	69
Figure 62 OpenID Configuration	70
Figure 63 nextcloud login page.....	71
Figure 64 Login to Nextcloud with Keycloak.....	71
Figure 65 Welcome Page to Nextcloud.	72

General Introduction

In the modern landscape of software development and IT operations, DevOps and Identity and Access Management (IAM) have become crucial pillars for achieving operational excellence and security. DevOps practices streamline development and operations, enabling continuous integration, delivery, and deployment, which accelerates time-to-market and enhances collaboration between development and operations teams. Meanwhile, IAM systems play a critical role in managing user identities and access permissions, ensuring secure and efficient access to resources across an organization.

Historically, the evolution of DevOps has transformed how software is developed and deployed. From the early days of siloed development and operations teams to the current model of integrated workflows, DevOps has driven significant improvements in software delivery speed and reliability. Concurrently, IAM has evolved from basic user authentication methods to comprehensive systems that manage identities, roles, and access across diverse IT environments. This evolution reflects the growing need for robust security and efficient access control in increasingly complex digital ecosystems.

Chapter 1: Project Foundation and Strategy.

1.1 Introduction

This first chapter is dedicated to presenting the general context of the project. We start by introducing Linsoft, the company where the internship took place. Then, we review the motivations behind this work before outlining the solution we have adopted. Following this, we explore key concepts related to high availability and Kubernetes and discuss the fundamental principles underlying our Keycloak deployment project.

1.2 Presentation of LinSoft

LinSoft is an IT engineering services company (SSII) specializing in IT solution integration in complex and heterogeneous technical environments. With three subsidiaries based in Tunis, Casablanca, and Algiers, LinSoft has emerged as a leader in IT solution integration and training in the Africa and Middle East (MEA) region.

Key Activities:

- **IT Solution Integration:** LinSoft offers tailored IT solution integration services for diverse technical environments, meeting specific client needs across the MEA region.
- **Training:** In addition to integration services, LinSoft provides specialized training programs to enhance IT skills within client organizations.

Locations: LinSoft strategically operates subsidiaries in Tunis, Casablanca, and Algiers, enabling effective service delivery to a broad clientele across the MEA region. Each location is equipped to address local needs while benefiting from LinSoft's global support.

Expertise and Innovation: LinSoft excels in deep expertise and overcoming challenges presented by complex IT environments. Leveraging cutting-edge technologies and

industry best practices, LinSoft is a trusted partner helping clients achieve their strategic objectives.

Vision: LinSoft aims to continue innovating and expanding capabilities to remain at the forefront of IT solution integration in the MEA region. The company is committed to delivering high-quality solutions, promoting operational excellence, and supporting digital growth for its clients.



FIGURE 1 LOGO OF LINSOFT.

1.3 Project Overview and Solution Approach

1.3.1 Problem Statement

Deploying Keycloak in a high-availability setup within a Kubernetes cluster involves several critical challenges. Keycloak, being a complex identity and access management system, needs to be highly available and resilient to ensure uninterrupted authentication and authorization services.

1.3.2 Objectives

The objectives of this project are as follows:

Implement High Availability for Keycloak: Develop a highly available Keycloak setup within a Kubernetes K3s cluster, ensuring that the identity and access management service remains operational even in the event of node failures.

Optimize Communication Between Services: Facilitate seamless integration and communication between Keycloak, Infinispan, and Nextcloud, ensuring that each service can interact efficiently within the Kubernetes environment.

Enhance Scalability: Implement auto-scaling mechanisms for Keycloak and associated services to ensure that the system can dynamically adjust to varying workloads, maintaining optimal performance at all times.

Ensure Resilience: Set up robust load balancing and failover strategies to guarantee that the system remains resilient in the face of disruptions, minimizing downtime and ensuring continuous availability.

Validate the System with Nextcloud Testing: Utilize Nextcloud as a testing ground to validate the high availability setup, ensuring that the integrated services perform as expected under various conditions.

1.3.3 State of the Art

In the past, managing identity and access in high availability (HA) environments was a complex and error-prone task. Early systems often required custom-built solutions that lacked scalability and security, resulting in manual configurations and vulnerabilities.

Keycloak has transformed this landscape by providing robust, built-in HA features such as clustering and integration with modern technologies like Kubernetes and Infinispan. This advancement simplifies the deployment of secure and scalable IAM solutions, moving away from cumbersome setups to a more efficient and resilient approach.

1.3.4 Our Solution

In response to the complexities and limitations of early identity and access management (IAM) systems, our solution leverages modern technologies to create a secure and high-availability setup. By deploying Keycloak with its built-in clustering capabilities, we address the scalability and reliability challenges those traditional systems struggled with. Our approach integrates Keycloak with Kubernetes (k3s) for streamlined container orchestration, Infinispan for efficient distributed caching, and Nextcloud for secure file management. This cohesive solution modernizes IAM by combining advanced features and technologies to ensure a robust, scalable, and secure environment.

1.4 Tools and Technologies

Our project employs several essential tools to create a secure and high-availability identity and access management solution. **Keycloak** serves as the core IAM platform, providing comprehensive features for authentication and authorization, with built-in clustering to ensure high availability. **Kubernetes (k3s)** is used for container orchestration, managing the deployment, scaling, and operations of Keycloak and associated services efficiently. **Infinispan** enhances system performance through distributed caching, reducing database load and improving response times. **Nextcloud** integrates with Keycloak to offer secure file synchronization and sharing capabilities.

These tools are integral to our solution, addressing various aspects of security, performance, and scalability. In the next chapter, we will delve into the specifics of each tool, exploring their configurations and the roles they play in our high-availability setup.

1.5 Methodology

TABLE 1 COMPARISONS OF SCRUM AGILE AND OTHER METHODOLOGIES

Aspect	Scrum Agile	Waterfall	Kanban
Approach	Iterative, incremental	Linear, sequential	Continuous flow
Flexibility	High, adaptable	Low, rigid phases	Medium, adaptive
Phases	Sprints (2-4 weeks)	Fixed stages (e.g., design, development)	No fixed phases
Customer Involvement	High, ongoing feedback	Low, initial and final stages	High, regular updates
Delivery	Frequent, incremental	One-time final delivery	Continuous, task-based
Documentation	Minimal, just-in-time	Extensive, detailed	Minimal, visual tracking

To effectively manage the development and deployment of our high availability (HA) solution, we adopted the Scrum methodology. Scrum is an agile framework designed to facilitate iterative progress, enhance collaboration, and adapt to changing requirements.

1.5.1 Why SCRUM ?

- **Flexibility:** Scrum's iterative approach allows for continuous refinement of the project, adapting to new insights and requirements as they arise.
- **Collaboration:** Regular Scrum ceremonies, such as daily standups and sprint reviews, promote effective communication and teamwork, ensuring alignment and addressing issues promptly.
- **Transparency:** Scrum provides visibility into the project's progress through sprint planning and reviews, helping stakeholders stay informed and make data-driven decisions.

1.5.2 Key Scrum Practices

- **Sprints:** We organized the project into short, time-boxed iterations (sprints) to deliver incremental improvements and gather feedback.
- **Daily Standups:** Brief daily meetings to discuss progress, challenges, and plans, keeping the team synchronized and focused.
- **Sprint Reviews and Retrospectives:** At the end of each sprint, we review completed work and reflect on processes to identify areas for improvement.

By employing Scrum, we ensure that our project remains adaptable, collaborative, and focused on delivering a high-quality, HA solution for Keycloak.



FIGURE 2 SCRUM DEVELOPMENT PROCESS

1.6 Conclusion

This chapter outlined the internship organization, identified the challenges in high availability and secure identity management, and introduced Keycloak as our solution. We discussed the tools and technologies used, including Kubernetes, Infinispan, and Nextcloud, and explained our adoption of Scrum for project management. In the next chapter, we will delve into the detailed configurations and roles of these tools.

Chapter 2: Architecture and Technologies Used.

2.1 Introduction

In this chapter, we will explore the key technologies and tools used in our project. We will define each component, discuss its purpose, and explain why it was chosen. This overview will set the stage for understanding their roles and contributions to our high-availability Keycloak setup.

2.2 DevOps Concept

2.2.1 DevOps Philosophy

High Availability is a crucial aspect of DevOps, ensuring reliable and efficient software delivery.

It's essential for several reasons:

Collaboration and Communication

DevOps is a software development approach that promotes collaboration, communication, and integration between development (Dev) and IT operations (Ops) teams. The goal is to deliver high-quality software products quickly and efficiently by breaking down silos between these teams.

Culture of Continuous Improvement

The underlying philosophy of DevOps encourages a culture of continuous improvement. Teams work together to deliver value to customers by regularly releasing software updates in small increments, allowing for frequent feedback and necessary adjustments to meet user needs.

Automation and Monitoring Tools

DevOps involves the adoption of automation and monitoring tools to optimize software delivery processes. These tools help streamline tasks, reduce errors, and enhance the overall efficiency of the development and deployment process.

Observability

Another important pillar of the DevOps philosophy is observability, which includes the ability to understand and diagnose complex systems, such as software applications and infrastructure. This is achieved by collecting and analyzing data from various sources, including logs, metrics, and traces.

Culture of Trust and Collaboration

In summary, DevOps aims to establish a culture of trust, collaboration, and continuous improvement within teams. All members work together to deliver high-quality software that effectively meets the needs of end users.

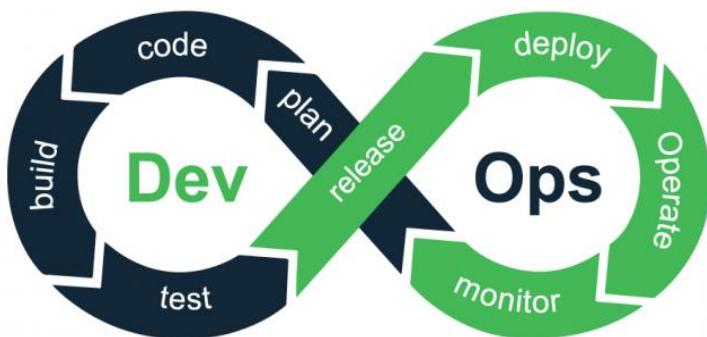


FIGURE 3 DEVOPS WORKFLOW PROGRESSION

2.2.2 Containerization

Containerization is an innovative method for packaging and running applications. Instead of installing and configuring each dependency on every machine, containerization allows everything to be bundled into a lightweight and portable "container." This means that the same container can run on any infrastructure without modification. By using containers, developers can create and deploy applications more quickly and securely.

Containers provide an elegant solution to the problem of transferring code between different environments. By including all necessary dependencies within the container, this approach ensures perfect consistency throughout the different stages of an application's lifecycle. Since the emergence of Docker in 2013, this technology has seen tremendous success, becoming a benchmark standard in the software development industry.

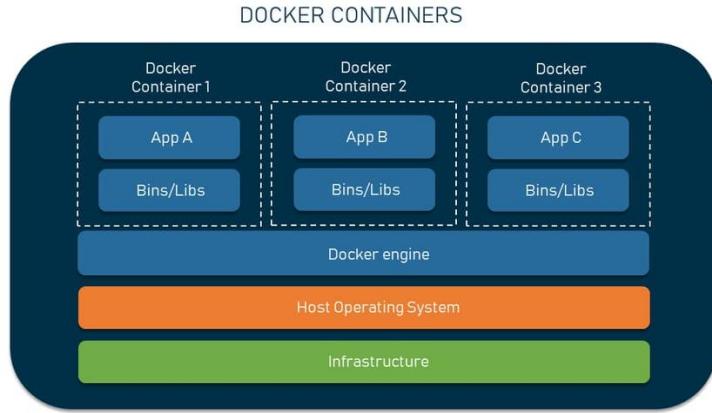


FIGURE 4 DOCKER CONTAINERS

2.2.3 Orchestration

With the rise of containers, the need for orchestrators such as Kubernetes (or K8s) has become evident. Although other container orchestration platforms exist, such as Docker Swarm and Apache Mesos, Kubernetes remains the most widely used and benefits from the support of a broad array of tools.

2.2.4 Kubernetes

Kubernetes is an open-source platform for container orchestration, originally developed by Google. It provides a framework for automating the deployment, scaling, and management of containerized applications. With Kubernetes, users can oversee and coordinate containers across multiple machines, offering a highly scalable and resilient infrastructure for running distributed applications.

2.2.5 Why Choose Kubernetes?

The major advantage of Kubernetes lies in its ability to allow users to focus on application functionality without worrying about implementation details. Through its abstractions, which manage groups of containers, Kubernetes separates the required behaviors from the components that provide them.

Kubernetes simplifies and automates many tasks. It facilitates the deployment of multi-container applications by reducing the effort needed to compose an application from a set of containers. The platform then handles the deployment and synchronization of components. It also simplifies scaling applications by adjusting them based on demand.

and optimizing the necessary resources. This scaling can be automated for improved efficiency.

Additionally, Kubernetes supports continuous deployment of new application versions, eliminating downtime for maintenance. It manages container networking, service discovery, and storage. Kubernetes can automate the provisioning and scaling of web servers based on traffic, adjusting instances up or down as needed. It also has advanced load balancing capabilities to optimize traffic routing.

By orchestrating containers across multiple hosts, Kubernetes optimizes hardware utilization to maximize resources available for applications. It controls and automates deployments and updates, while managing services declaratively to ensure execution according to specifications. Kubernetes monitors application health and automatically repairs issues through placement, startup, replication, and automatic scaling.

Finally, Kubernetes plays a crucial role in container management, particularly in public cloud environments where efficient resource management is essential to avoid orphaned or under-provisioned machines. In summary, Kubernetes excels in orchestration by optimizing and automating the management of containerized applications.

2.2.6 Kubernetes Architecture

The architecture of Kubernetes, as shown in Figure 1.5, is based on a master node and worker node model, with each having specific roles to ensure the proper functioning and resilience of the cluster. Master nodes are responsible for overall cluster management, including task orchestration, resource scheduling, and maintaining the desired state of applications. Worker nodes, on the other hand, run containerized applications and services according to the instructions and configurations defined by the master nodes.

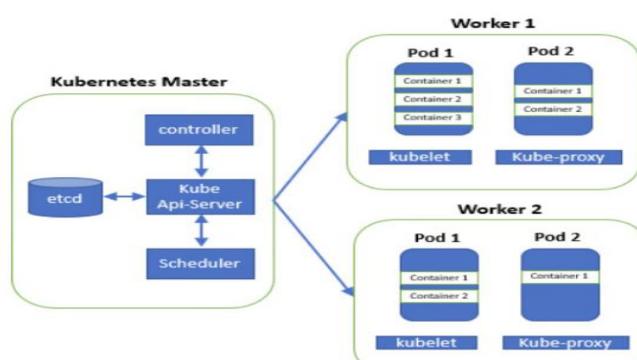


FIGURE 5 KUBERNETES ARCHITECTURE

- **Master Node:** This is the cluster manager and the entry point for all administrative tasks. It includes several key components:
 - **kube-apiserver:** This is the main interface for managing the cluster. It validates and executes user requests.
 - **etcd:** This is the key-value database for the Kubernetes cluster, providing persistent storage for all container data.
 - **kube-scheduler:** It assigns tasks to worker nodes based on resource usage and specific needs.
 - **kube-controller-manager:** It monitors and maintains the desired state of the cluster, taking corrective actions as necessary.
- **Worker Node:** This is the server that runs applications and is controlled by the master node. It includes the following components:
 - **Kubelet:** This is an agent running on each worker node, managing pods and containers, and ensuring their proper operation.
 - **Kube-proxy:** This is a network proxy that configures network rules to enable communication between cluster services.
 - **Container Runtime:** This is the execution environment for containers, such as Docker, which manages the lifecycle of containers.

2.2.7 Kubernetes Objects

Kubernetes aims to abstract the complexity of managing a container fleet, and to achieve this, several objects exist, including:

- **Node:** A worker machine in the Kubernetes cluster. These machines can be physical, virtual, or cloud instances, and they run the containers that make up your applications.
- **Pod:** The smallest unit in Kubernetes, encompassing one or more containers that share the same network and storage space. Each pod has a unique IP address and a shared volume, allowing containers to easily share data.
- **Replicas:** The number of instances of a pod, enabling horizontal scaling of applications.

- **ReplicaSet:** Ensures that the specified number of pod replicas are always active, continuously monitoring the state of pods and automatically adjusting the number of replicas as needed.
- **Deployment:** Defines the desired state of the deployed application and provides declarative means to update pods and ReplicaSets.
- **Service:** A logical set of pods exposed as a network service, providing a unique IP address and DNS name for a group of pods. Services also manage load balancing configuration.
- **Endpoint:** Represents the IP address and port of a service, automatically created when a service is created with the corresponding pods.
- **Volumes:** Kubernetes volumes provide persistent storage for pods, unlike the default ephemeral storage. They allow data sharing between containers within a pod and can be mounted at specific points within the container. This offers a solution for durable storage and data sharing between containers.
- **Namespaces:** Kubernetes provides resource partitioning into non-overlapping sets called "namespaces." They are designed for use in environments with many users across multiple teams or projects, or even to separate environments such as development, testing, and production.

2.2.8 Kubernetes Distributions

Kubernetes offers the option to choose from several distributions, including installers that may be limited to certain cloud service providers but can also be used in bare metal or private cloud environments. Table 1.1 provides a comparative summary of the main Kubernetes distributions.

TABLE 2 KUBERNETES DISTRIBUTIONS.

Détails	Kubeadm	K3s	Kubespray
Installer	Official Kubernetes installer	Lightweight Kubernetes with automated installer	Kubernetes based on an Ansible installation
Rôle	SRE / DevOps	SRE / DevOps, IoT	SRE / DevOps

Ecosystem	Kubernetes Upstream	Rancher Kubernetes Upstream	Kubernetes Upstream
Infrastructure	Multi-platform	Virtual machine	Multi-platform
Ingress and Storage Control	No	Yes	Yes

After a thorough analysis of all the Kubernetes distributions presented and considering our specific needs, we have decided to choose K3S as the ideal solution for our project. K3S offers a lightweight approach with an automated installer, perfectly aligning with our requirements for deploying and managing Kubernetes clusters.

2.3 What is High Availability?

High availability refers to the design and implementation of systems that aim to minimize downtime by ensuring continuous operation, even in the face of hardware failures, software bugs, or other unexpected issues. Achieving high availability typically involves redundancy, failover mechanisms, and load balancing to ensure that if one component fails, another can immediately take over without affecting the end user's experience.

2.3.1 Why is High Availability Important?

High availability is essential for several reasons:

- **Business Continuity:** In today's digital world, businesses rely heavily on their IT systems. Downtime can disrupt operations, leading to financial losses and damage to customer trust.
- **User Experience:** For services that are user-facing, any downtime can lead to frustration and dissatisfaction among users, potentially leading to loss of customers.
- **Compliance and SLA Requirements:** Many industries have strict regulations and Service Level Agreements (SLAs) that require a certain level of uptime. High availability ensures compliance with these standards.

- **Global Operations:** For organizations operating globally, services must be available 24/7 to accommodate different time zones. High availability supports this need by ensuring that systems remain operational around the clock.

2.3.2 High Availability in Our Project

In the context of our project, high availability is paramount due to the nature of the services we are deploying, particularly Keycloak as an identity and access management solution. In a Kubernetes K3s cluster, ensuring the high availability of Keycloak, alongside Infinispan for caching and Nextcloud for testing, is crucial. Any downtime in these services could lead to significant disruptions in user authentication, data caching, and collaboration, which are critical for the overall functionality and performance of the system.

By implementing high availability, we ensure that:

- **Keycloak** remains operational, providing continuous authentication and authorization services.
- **Infinispan** ensures data consistency and quick access to cached data, even during high loads.
- **Nextcloud** maintains its functionality for file sharing and collaboration during testing, contributing to the overall reliability of the system.

2.4 What is Keycloak?

2.4.1 What is Identity and Access Management (IAM)?

Identity and access management (IAM) is a cybersecurity framework in which the IT team controls access to computer systems, cloud applications, networks, and assets based on each user or device's digital identity.

An IAM tool's core functions are to:

1. **Assign a single digital identity to each user**
2. **Authenticate the user**
3. **Authorize appropriate access to relevant resources**

4. Monitor and manage identities to align with changes within the organization

2.4.2 IAM benefits

When organizations implement an identity and access management solution, they get to enjoy the following benefits:

Optimal access and authentication customized for individual entities

Improved productivity by providing users with SSO solutions that prevent them from having to memorize multiple passwords

Reduced risk of data breaches thanks to the right users having the right amount of access to the right assets.

Increased collaboration among different teams and vendors because security is implemented throughout all processes.

Compliance regulations and standards baked into the tool.

2.4.3 Introducing Keycloak



FIGURE 6 LOGO OF KEYCLOAK

Keycloak is an **open-source Identity and Access Management tool** focused on modern applications such as single-page applications, mobile apps, and REST APIs. The project began in 2014 with a strong emphasis on simplifying application security for developers. Since then, it has become a well-established open-source project with a robust community and diverse user base, supporting a range of scenarios from small websites with few users to large enterprises with millions of users.

Keycloak offers customizable login pages with support for strong authentication, password recovery, password updates, and acceptance of terms and conditions, all

without requiring additional coding. Its pages support custom themes, allowing easy integration with corporate branding and existing applications.

By centralizing authentication with Keycloak, applications do not need to manage various authentication mechanisms or securely store passwords. Instead, Keycloak provides security tokens that grant only the necessary access.

Keycloak supports a wide range of authentication factors, including **Multi-Factor Authentication (MFA)** and **Strong Authentication (SA)**, with options like **OTPs**, security devices, WebAuthn, and passwords. It also offers **single sign-on (SSO)** and session management, enabling users to access multiple applications with a single login. Both users and administrators have full visibility into active sessions and can terminate them remotely if needed.

Keycloak builds on industry-standard protocols such as **OAuth 2.0**, **OpenID Connect**, and **SAML 2.0**, facilitating integration with both new and existing applications. It includes its own user database for easy setup but can also integrate with existing identity infrastructure, including social networks, enterprise identity providers, and directories like **Active Directory** and **LDAP**.

Keycloak is lightweight, easy to install, highly scalable, and provides high availability through clustering. It supports a range of use cases out of the box but is also highly customizable and extendable, with numerous extension points for custom authentication mechanisms, user store integrations, and token manipulation. Custom login protocols can also be implemented.

2.4.4 Overview of Keycloak and Other IAM Solutions

TABLE 3 OVERVIEW OF KEYCLOAK AND OTHER IAM SOLUTIONS

	keycloak	Okta	Auth0	Azure Active Directory (AD)
Open source	Yes	No	No	No
Customizable login page	Yes	Limited	Yes	Limited
Multi-Factor Authentication(MFA)	Yes	Yes	Yes	Yes
Single Sign-On(SSO)	Yes	Yes	Yes	Yes
Industry Standards support	OAuth 2.0, OpenID Connect, SAML 2.0	OAuth 2.0, OpenID Connect, SAML 2.0	OAuth 2.0, OpenID Connect, SAML 2.0	OAuth 2.0, OpenID Connect, SAML 2.0
User directory integration	Yes (LDAP, Active Directory, social networks)	Limited	Yes (social networks, enterprise)	Yes (Active Directory)
Scalability	High (Clustering capabilities)	High (Cloud-based)	High (Cloud-based)	High (Cloud-based)
Compliance	GDPR, HIPAA	GDPR, HIPAA, others	GDPR, HIPAA, others	GDPR, HIPAA, others
Cost	Free	Subscription-based	Subscription-based	Subscription-based

2.4.5 Why Choose KEYCLOAK ?

In selecting an Identity and Access Management (IAM) solution for our project, we considered several options available in the market. Keycloak stood out as the optimal choice for the following reasons:

- **Open-Source and Cost-Effective:** Keycloak is an open-source solution, eliminating the need for costly licensing fees while providing a robust feature set comparable to proprietary alternatives.

- **Comprehensive and Flexible Features:** Keycloak offers essential IAM capabilities like Single Sign-On (SSO), Multi-Factor Authentication (MFA), and support for OAuth 2.0, OpenID Connect, and SAML 2.0 protocols. It also allows seamless integration with existing identity providers, including LDAP and Active Directory.
- **Customizability:** The platform provides extensive customization options, enabling us to tailor authentication flows, user interfaces, and security measures to our specific project needs without requiring significant changes to our applications.
- **Scalability and High Availability:** Keycloak supports clustering and high availability configurations, making it suitable for both small-scale applications and large enterprises with millions of users. This aligns perfectly with our project's requirements for a scalable and resilient IAM solution.
- **Strong Community Support:** As a mature open-source project, Keycloak benefits from a vibrant community and comprehensive documentation, facilitating both development and problem-solving processes.
- **Enhanced Security:** Keycloak ensures the security of our applications by securely managing user credentials, providing token-based authentication, and offering fine-grained access controls, all crucial for maintaining the integrity of our system.

These advantages make Keycloak the ideal IAM solution for our project, providing a balance of functionality, flexibility, and security that meets our requirements effectively.

2.4.6 Installing and running Keycloak

Keycloak provides a few options on how it can be installed, including the following:

- Running as a container on Docker
- Installing and running Keycloak locally (which will require a Java virtual machine, such as OpenJDK)
- Running Keycloak on Kubernetes
 - Using the Keycloak Kubernetes Operator

If you already have Docker installed on your workstation, this is the recommended approach as it simplifies getting started with Keycloak. If Docker is not installed, you can begin by setting it up locally. The only additional dependency required is a Java virtual machine. Keycloak can also be deployed to Kubernetes, where you have the option of using the Keycloak Kubernetes Operator. This operator streamlines installation, configuration, and management. In our project, we will provide detailed instructions for deploying the Keycloak Operator in a Kubernetes cluster.

2.4.7 Discovering the Keycloak admin and account consoles

The Keycloak admin console provides an extensive and friendly interface for administrators and developers to configure and manage Keycloak.

To access the admin console, open http://ip_address:8080/admin in a browser. You will be redirected to the Keycloak login page, where you can log in with the admin username and password you created in the previous section while installing Keycloak.

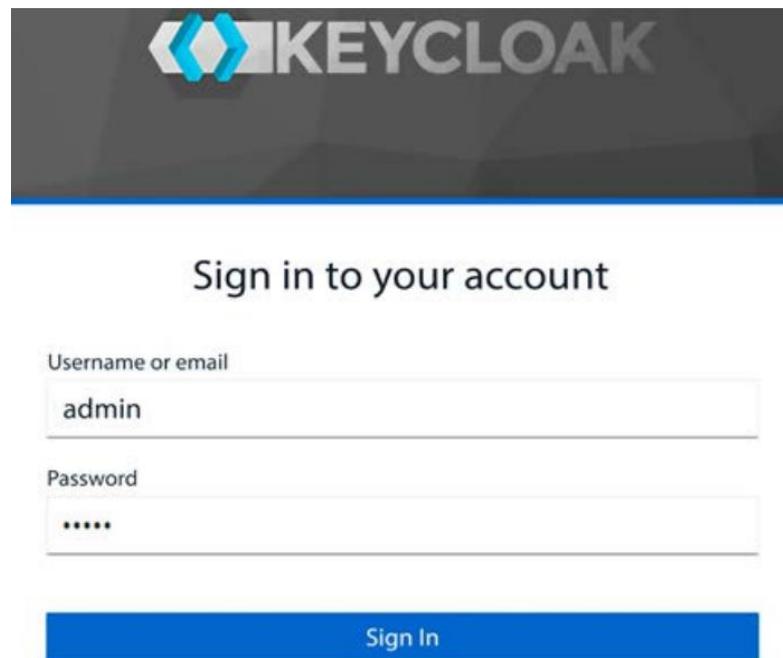


FIGURE 7 KEYCLOAK LOGIN INTERFACE

2.4.7.1 Key Concepts in Keycloak

Realm : A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

Groups: Groups manage groups of users. Attributes can be defined for a group. You can map roles to a group as well. Users that become members of a group inherit the attributes and role mappings that group defines.

Clients: Clients are entities that can request Keycloak to authenticate a user. Most often, clients are applications and services that want to use Keycloak to secure themselves and provide a single sign-on solution. Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Keycloak.

Roles: Roles identify a type or category of user. Admin, user, manager, and employee are all typical roles that may exist in an organization. Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine-grained and hard to manage.

Credentials: Credentials are pieces of data that Keycloak uses to verify the identity of a user. Some examples are passwords, one-time-passwords, digital certificates, or even fingerprints.

2.5 What is INFINISPAN ?



FIGURE 8 LOGO OF INFINISPAN

Infinispan is a highly scalable, distributed, and open-source key/value data store and data grid platform written in Java. It is designed to handle large-scale, highly concurrent data operations across modern multi-processor and multi-core architectures. Infinispan is

commonly used as a distributed cache, providing fast, in-memory access to data, but it can also function as a NoSQL key/value store or an object database, offering flexibility in how data is stored and retrieved.

Infinispan offers features that are essential for modern, data-intensive applications:

- **Distributed Caching:** Infinispan provides a high-performance distributed cache that allows for rapid data retrieval and storage across multiple nodes.
- **Data Grid Capabilities:** It serves as a data grid platform, enabling large-scale data operations with high concurrency and low latency.
- **Persistence Options:** Infinispan supports data persistence to ensure data durability and recovery, unlike some caching solutions that only provide in-memory storage.
- **Advanced Features:** The platform includes features such as near caching, transactions, and integrations with other systems, enhancing its functionality beyond simple caching.

TABLE 4 COMPARING INFINISPAN WITH OTHER SOLUTIONS

Feature	Infinispan	Redis	Memcached	Apache Cassandra
Open source	Yes	Yes	Yes	Yes
Distributed cache	Yes	Yes	Yes	No
Data persistence	Yes	No	No	Yes
Advanced Features	Yes	Limited	Limited	Extensive
Java integration	Yes	No	No	No
Scalability	High	High	High	High

2.5.1 Why Choose Infinispan?

When selecting a data management solution for our project, Infinispan emerged as a top choice due to several compelling factors:

- **Scalability and Performance:** Infinispan's architecture supports horizontal scaling and high concurrency, making it suitable for large-scale applications with high data demands.
- **Data Durability:** Unlike some in-memory-only caches, Infinispan provides options for data persistence, ensuring data integrity and recovery.
- **Java Compatibility:** As a Java-based solution, Infinispan integrates seamlessly with Java applications, aligning with our technology stack.
- **Flexibility:** Infinispan's ability to function both as a distributed cache and a NoSQL data store offers flexibility in data management strategies.
- **Community and Support:** Infinispan benefits from a strong open-source community and comprehensive documentation, aiding in deployment and troubleshooting.

2.5.2 Installing and Running Infinispan

Infinispan can be installed and run using several methods:

- **Running as a Docker Container:** This approach simplifies deployment and management, especially in development environments.
- **Standalone Installation:** For local setups, Infinispan can be installed directly on a machine with Java Virtual Machine (JVM) support.
- **Kubernetes Deployment:** For scalable and resilient environments, Infinispan can be deployed on Kubernetes using the Infinispan Operator, which manages deployment, scaling, and configuration.

2.6 What is Nextcloud ?



FIGURE 9 LOGO OF NEXTCLOUD

2.6.1 Introducing Nextcloud

Nextcloud is a powerful, open-source platform for file synchronization and sharing, designed to offer a robust and secure alternative to commercial cloud storage solutions. Since its initial release in 2016, Nextcloud has been adopted by organizations and individuals who seek to maintain control over their data while benefiting from a rich set of collaborative features.

Nextcloud provides a range of functionalities tailored to modern collaboration and file management needs:

- **File Synchronization and Sharing:** Nextcloud allows users to synchronize files across multiple devices and share them with others, both inside and outside their organization.
- **Collaboration Tools:** It includes integrated tools for real-time collaboration, such as document editing, chat, and calendar management, enhancing productivity.
- **Security and Privacy:** Nextcloud emphasizes strong security measures, including end-to-end encryption, two-factor authentication, and detailed access controls, ensuring that data remains secure and private.
- **Customization and Integration:** The platform supports numerous plugins and integrations, allowing users to extend its capabilities and integrate with other systems.

Nextcloud is designed to be scalable and adaptable, suitable for personal use as well as enterprise environments. Its open-source nature means that users can customize and extend the platform according to their specific needs.

2.6.2 Comparing Nextcloud with Other Solutions

Feature	Nextcloud	Dropbox	Google Drive	OneDrive
Open Source	Yes	No	No	No
File Synchronization	Yes	Yes	Yes	Yes
Collaboration Tools	Yes	Limited	Yes	Yes
End-to-End Encryption	Yes	No	No	No
Customization	Extensive	Limited	Limited	Limited
Integration Options	Extensive	Limited	Extensive	Extensive
Scalability	High	Medium	High	High

TABLE 5 COMPARING NEXTCLOUD WITH OTHER SOLUTIONS

2.6.3 Why Choose Nextcloud?

In choosing a file synchronization and collaboration solution for our project, Nextcloud emerged as the preferred option due to several key benefits:

- **Open-Source Flexibility:** Nextcloud's open-source nature allows for extensive customization and control over the platform, avoiding vendor lock-in.
- **Strong Security Features:** The platform offers robust security features, including end-to-end encryption and advanced access controls, ensuring data privacy and protection.
- **Comprehensive Collaboration Tools:** Nextcloud provides a suite of collaboration tools, integrating file sharing, document editing, and calendar management into a single platform.

- **Customizability and Integration:** The ability to customize and integrate with other applications makes Nextcloud a versatile choice for meeting specific organizational needs.
- **Community and Support:** Nextcloud benefits from an active community and strong support, providing resources and assistance for implementation and troubleshooting.

2.6.4 Installing and Running Nextcloud

Nextcloud can be installed and operated through several methods:

- **Docker Container:** Using Docker simplifies deployment and management, especially in development and testing environments.
- **Manual Installation:** For more control, Nextcloud can be manually installed on a server, requiring a web server, database, and PHP setup.
- **Kubernetes Deployment:** For scalable and resilient deployment, Nextcloud can be run on Kubernetes, with Helm charts available for simplified setup and management.

In our project, we opted to deploy Nextcloud using Docker containers to facilitate easy setup and integration with our existing infrastructure.

2.3 Conception and Architecture of Our Project

2.3.1 K3s Architecture with High Availability Setup

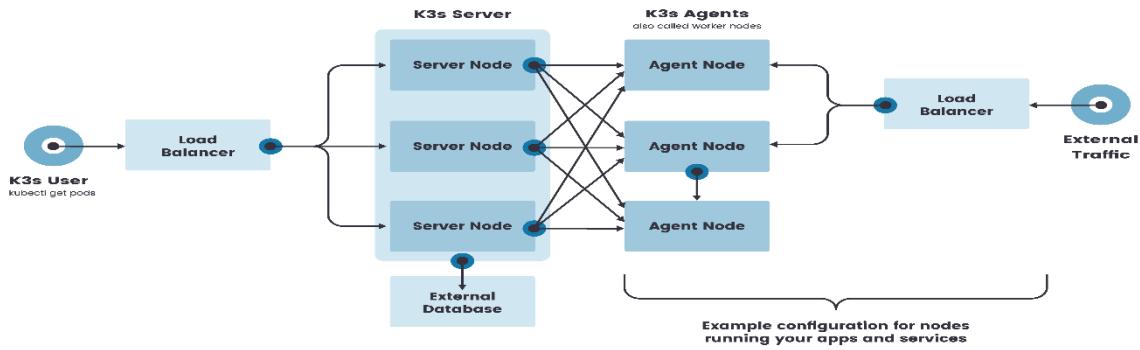


FIGURE 10 K3S ARCHITECTURE

In a high availability (HA) setup of K3s, the architecture is designed to ensure both reliability and fault tolerance. This setup involves two master nodes and two worker nodes:

1. **Master Nodes:** There are two master nodes responsible for managing the Kubernetes control plane. They handle critical tasks such as scheduling, API requests, and maintaining cluster state. By having two master nodes, the system ensures that if one master fails, the other can continue to manage the cluster, thus preventing a single point of failure and enhancing the cluster's reliability.
2. **Worker Nodes:** The two worker nodes run the application workloads and services. These nodes are responsible for executing the containers scheduled by the master nodes. In this HA setup, the workload is distributed across both worker nodes to balance resource usage and provide redundancy. If one worker node fails, the remaining node continues to operate, maintaining the availability of the applications running on the cluster.

This architecture is designed to provide continuous availability and robust performance, making it suitable for both development and production environments.

2.3.2 Multi-Node Deployment of Keycloak, Infinispan, and PostgreSQL in Kubernetes

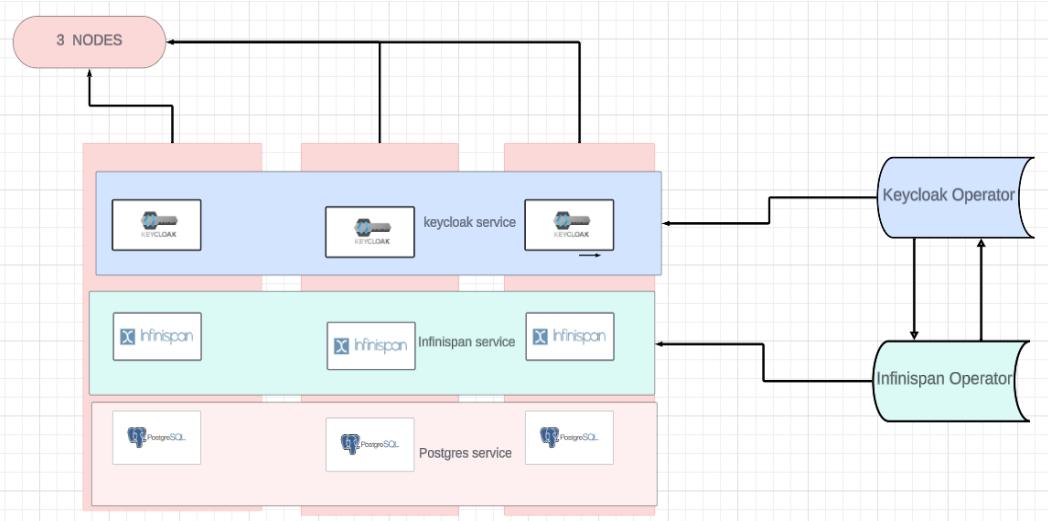


FIGURE 11 MULTI-NODE DEPLOYMENT OF KEYCLOAK, INFINISPAN, AND POSTGRESQL IN KUBERNETES

The diagram illustrates a Kubernetes-based deployment architecture that involves multiple nodes and services, focusing on Keycloak, Infinispan, and PostgreSQL, managed by their respective operators.

Components :

1. 3 Nodes:

- This represents the three Kubernetes nodes (which could be physical or virtual machines) that make up the cluster. These nodes host the various services (Keycloak, Infinispan, and PostgreSQL) and allow for high availability and scalability.

2. Keycloak Service :

- The Keycloak service is shown to be deployed across multiple pods (three instances) within the Kubernetes cluster. Keycloak provides identity and access management, handling user authentication and authorization for applications.
- These instances are managed by the **Keycloak Operator**, which is responsible for deploying, managing, and scaling Keycloak instances.

3. Infinispan Service :

- Infinispan is deployed as a distributed in-memory data grid across multiple pods (three instances). It provides caching and data storage capabilities to improve performance and reliability.
- The **Infinispan Operator** manages these Infinispan instances, handling tasks such as scaling, configuration, and ensuring high availability.

4. Postgres Service :

- PostgreSQL is deployed as a database service across multiple pods (three instances) to provide persistent storage for the Keycloak service and potentially other applications.
- While not explicitly managed by an operator in the diagram, PostgreSQL is a critical component providing relational data storage.

5. Operators:

- **Keycloak Operator:** Manages the Keycloak service, ensuring that the Keycloak pods are properly deployed, configured, and scaled. It interacts with the Kubernetes API to maintain the desired state of the Keycloak instances.
- **Infinispan Operator:** Similarly, this operator manages the Infinispan service, automating the deployment and management of the Infinispan pods within the Kubernetes cluster.

Workflow :

- The three nodes of the Kubernetes cluster collectively run the services: Keycloak, Infinispan, and PostgreSQL. Each service is deployed in a replicated manner to ensure high availability and fault tolerance.

- The **Keycloak Operator** manages the lifecycle of the Keycloak service, ensuring that the required number of instances are running and are correctly configured.
- The **Infinispan Operator** manages the Infinispan service in a similar fashion, maintaining the integrity and availability of the distributed data grid.
- PostgreSQL runs as a database service that supports the Keycloak service, storing user data, session information, and other relevant data.

This architecture ensures a robust and scalable deployment of Keycloak for identity management, Infinispan for caching and data grid functionality, and PostgreSQL for persistent storage, all within a Kubernetes cluster. The use of operators (Keycloak and Infinispan) automates the management of these services, facilitating easier maintenance and scaling in a production environment.

2.3.3 Nextcloud and Keycloak Integration in a k3s Cluster

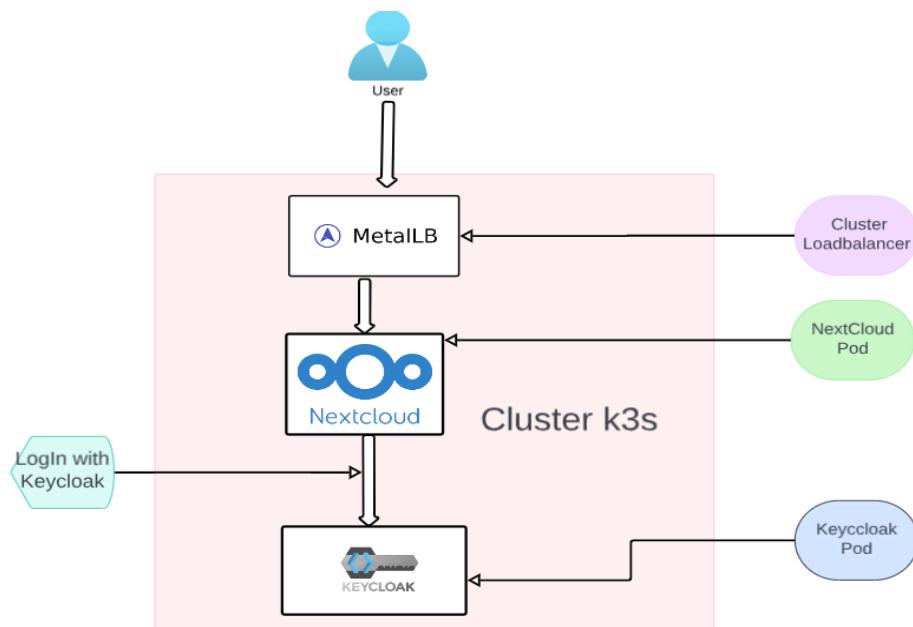


FIGURE 12 NEXTCLOUD AND KEYCLOAK INTEGRATION IN A K3S CLUSTER

The diagram represents a high-level architecture for integrating Nextcloud with Keycloak for user authentication within a Kubernetes k3s cluster. Here's a detailed description:

Components:

- **User:** Represents an end-user who interacts with the system.
- **MetalLB:** A load balancer implementation for Kubernetes, which handles external traffic and routes it to the appropriate services within the cluster. In this context, MetalLB distributes incoming traffic to Nextcloud and Keycloak services within the cluster.
- **Nextcloud:** A popular open-source file-sharing and collaboration platform. In this setup, it's deployed within a pod in the k3s cluster. Users access Nextcloud via MetalLB.
- **Keycloak:** An open-source identity and access management solution. It's used here for handling authentication and login processes for Nextcloud. Keycloak is also deployed within a pod in the k3s cluster.
- **Cluster Load Balancer:** Likely represents an external or internal load balancer in the cluster, working with MetalLB to ensure high availability and traffic distribution across Nextcloud and Keycloak pods.

Workflow :

- **User Access:** The user initiates access to the Nextcloud service, and this request is routed through MetalLB.
- **Nextcloud Authentication:** When the user attempts to log in to Nextcloud, the login process is delegated to Keycloak.
- **Login with Keycloak:** Keycloak manages user authentication. If the user is successfully authenticated, Keycloak provides the necessary authentication tokens back to Nextcloud, allowing the user to access Nextcloud's features.
- **Pod Interaction:** Both Nextcloud and Keycloak are running within separate pods in the k3s cluster, ensuring that they can scale independently and maintain high availability.

Cluster k3s:

The entire setup is running on a k3s cluster, which is a lightweight Kubernetes distribution designed for running Kubernetes clusters in resource-constrained environments.

This architecture leverages the integration of Nextcloud and Keycloak within a k3s cluster to provide a scalable and secure environment where users can access cloud services with centralized authentication managed by Keycloak. MetalLB facilitates the load balancing of incoming traffic, ensuring that both services are available and responsive.

2.7 Conclusion

In summary, integrating Kubernetes (or k3s), Keycloak, and Infinispan offers a powerful solution for building scalable and secure IT environments. Kubernetes orchestrates and manages containerized applications efficiently, Keycloak provides robust identity and access management, and Infinispan enhances performance with distributed caching. Together, these technologies create a resilient and high-performing infrastructure capable of handling modern application demands.

Chapter 3: Overall Architecture and Implementation Strategy.

3.1 Introduction

In this chapter, we will guide you through the process of installing, configuring, and deploying a robust architecture that includes Keycloak, Infinispan, Nextcloud, and MySQL. This documentation provides step-by-step instructions to help you set up each component within a Kubernetes environment, ensuring a scalable, high-availability infrastructure that meets your organization's needs. Whether you are managing user identities, handling large data volumes, or facilitating secure file sharing, this chapter will equip you with the knowledge to successfully implement and deploy these technologies.

3.2 Work Environment

The technical framework used for the implementation of our project includes the following components:

- **Six virtual machines:** One machine dedicated to development, two master nodes, two worker nodes, one virtual machine serving as an HAProxy load balancer, and one virtual machine for an external database. These virtual machines are deployed on a local infrastructure, as detailed in Table below.
- **Hardware Resources:** The computer hosting the virtual machines must have at least 15 GB of RAM to ensure optimal performance.

Type of VM	RAM	vCPU	Hard Drive	Number of VMs	OS
Development Machine	4 GB	2	50GB	1	Ubuntu

Master node	3GB	2	30GB	2	Ubuntu
Worker node	4GB	2	35GB	2	Ubuntu
Load balancer	2GB	1	30GB	1	Ubuntu
Database	2GB	1	30GB	1	Ubuntu

TABLE 5 CHARACTERISTICS OF THE VIRTUAL MACHINES USED IN THE PROJECT

3.3 Infrastructure Setup

Before starting the installation of our Kubernetes cluster, it is essential to prepare the configuration of the machines involved in this operation. We have chosen to use Ubuntu Linux distribution, version 22.04, for this purpose.

We intend to deploy a highly available K3s architecture. To achieve this, we will implement the architecture as illustrated in Figure 13.

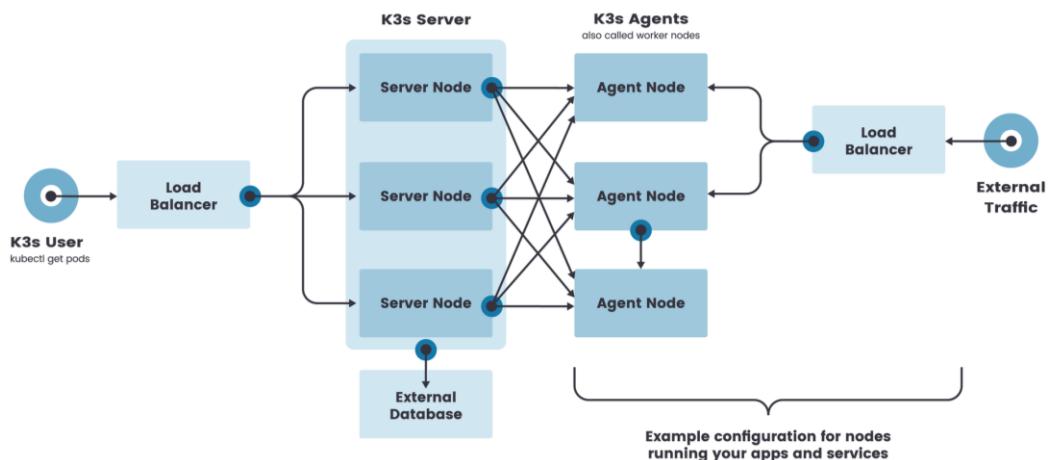


FIGURE 13 K3S ARCHITECTURE.

3.3.1 Configuration of a Load Balancer

Given that our deployment includes multiple Kubernetes master nodes, it is essential to have a load balancer to distribute traffic among the 3 nodes. In this configuration, we have chosen to use HAProxy as the load balancer. First, we begin by installing HAProxy.

```
$ sudo apt install haproxy
```

Next, we update the HAProxy configuration in the file **/etc/haproxy/haproxy.cfg**, incorporating the master node parameters as shown in Figure14.

This configuration defines a frontend named "kubernetes-frontend," listening on port **6443** and configured in TCP mode. Incoming requests are routed to the backend "**kubernetes-backend**." This backend is set up to load balance using the "roundrobin" mode among three servers representing the Kubernetes master nodes. Each server is specified with its **IP address** and port **6443** and is subject to periodic health checks.

```
frontend kubernetes-frontend
    bind *:6443
    mode tcp
    option tcplog
    timeout client 10s
    default_backend kubernetes-backend

backend kubernetes-backend
    timeout connect 10s
    timeout server 10s
    mode tcp
    option tcp-check
    balance roundrobin
    server k3s-master-1 192.168.40.139:6443 check fall 3 rise 2
    server k3s-master-2 192.168.40.140:6443 check fall 3 rise 2
```

FIGURE 14 HAProxy CONFIGURATION.

In case of a connection failure for a server, it is considered unavailable after three consecutive attempts but will be reintegrated after passing two consecutive health checks.

After making this configuration, it is necessary to restart HAProxy:

```
$ sudo systemctl restart haproxy
```

3.3.2 Configuration of the External Database

First, we need to install and configure Docker and Docker Compose on our virtual machine. Next, we create a **docker-compose** file, as shown in Figure . This file defines the MySQL database as well as a MySQL administration tool, Adminer, for managing the database.

```

GNU nano 6.2                               docker-compose.yaml
version: "3.3"
services:
  db:
    image: mysql
    restart: always
    environment:
      MYSQL_DATABASE: "keycloak"
      # So you don't have to use root, but you can if you like
      MYSQL_USER: "k3s"
      # You can use whatever password you like
      MYSQL_PASSWORD: "k3s"
      # Password for root access
      MYSQL_ROOT_PASSWORD: "root"
    ports:
      # <Port exposed> : < MySQL Port running inside container>
      - "3306:3306"
    expose:
      # Opens port 3306 on the container
      - "3306"
    # Where our data will be persisted
    volumes:
      - my-db:/var/lib/mysql
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
# Names our volume
volumes:
  my-db:

```

The terminal window shows the configuration of an external database using Docker Compose. The `docker-compose.yaml` file defines two services: `db` (MySQL) and `adminer` (Adminer web interface). The `db` service uses the `mysql` image, restarts always, and exposes port 3306. It also defines environment variables for the database, user, password, and root password. The `adminer` service uses the `adminer` image, restarts always, and exposes port 8080. A volume named `my-db` is used for persisting data.

FIGURE 15 CONFIGURATION OF THE EXTERNAL DATABASE

To start MySQL and Adminer, we execute the following command:

\$ docker-compose up -d

Finally, we review the images and containers that have been created, as shown in Figure16.

```

database@database-virtual-machine:~$ sudo docker images
[sudo] password for database:
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
adminer          latest   2f7580903a1d  5 weeks ago   250MB
mysql            latest   a82a8f162e18  5 weeks ago   586MB
database@database-virtual-machine:~$ sudo docker ps
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS
 NAMES
cf811c22a85e    mysql      "docker-entrypoint.s..."  2 days ago   Up 12 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
 database_db_1
01ce747401a7    adminer    "entrypoint.sh php -..."  3 days ago   Up 12 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
 database_adminer_1

```

FIGURE 16 LIST OF IMAGES

3.3.3 Setting Up the Cluster

In the process of creating our cluster, we configured three worker nodes and three master nodes to form our infrastructure.

3.3.3.1 Master Node Configuration

Before installing K3s, we need to store the database connection URL in a variable that will be automatically used during the installation process.

```
$ export K3S_DATASTORE_ENDPOINT='mysql://username:password@  
tcp(database_ip_or_hostname:port)/db'
```

Next, we use the following command to install K3s on the first master node:

The --node-taint parameter instructs Kubernetes that only critical pods should be run on our master nodes. Regular pods, such as custom applications, should be run on the worker nodes.

We need to adjust the permissions to run the k3s command on the master node without needing sudo:

```
$ sudo chmod 644 /etc/rancher/k3s/k3s.yaml
```

Once our first master node is configured, we proceed with the installation on the other master nodes. To do this, we first obtain the token needed to connect a node to the Kubernetes cluster.

```
$ sudo cat /var/lib/rancher/k3s/server/node-token
```

Next, we take the output from the previous command and, on each additional master node, use the following command to install K3s:

```
$ curl -sfL https://get.k3s.io | \ sh -s - server \ --token=token \ --  
datastore-endpoint="mysql://username:password@tcp  
(database_address:3306)/db" \ --node-taint  
CriticalAddonsOnly=true:NoExecute \ --tls-san load_balancer_ip:6443
```

After installation, we need to verify that K3s is running on the additional master node by checking its status, as shown in Figure17.

```
workernode1-virtual-machine  Ready    control-plane,master  2d1h    v1.30.4+k3s1
workernode2-virtual-machine  Ready    control-plane,master  2d      v1.30.4+k3s1
worker1@worker1-virtual-machine:~$
```

FIGURE 17 VERIFICATION OF MASTER NODES

3.3.3.2 Worker Node Configuration

Now that we have configured our master nodes, we can proceed with the installation of K3s on the worker nodes.

On each worker node, we use the same token that was used to configure the other master nodes and execute the following command.

```
curl -sfL https://get.k3s.io | K3S_URL=https://myserver:6443
K3S_TOKEN=mynodetoken sh -
```

Now that we have configured our master nodes, we can proceed with the installation of K3s on the worker nodes.

On each worker node, we use the same token as was used to configure the other master nodes and execute the command shown in Figure18 to install K3s.

```
worker1@worker1-virtual-machine:~$ kubectl get nodes
NAME           STATUS   ROLES          AGE   VERSION
worker1-virtual-machine  Ready    <none>        2d    v1.30.4+k3s1
worker2-virtual-machine  Ready    <none>        2d    v1.30.4+k3s1
workernode1-virtual-machine  Ready    control-plane,master  2d1h    v1.30.4+k3s1
workernode2-virtual-machine  Ready    control-plane,master  2d      v1.30.4+k3s1
worker1@worker1-virtual-machine:~$
```

FIGURE 18 VERIFICATION OF WORKER NODES

3.3.3.3 Configuration of the Development Machine

After successfully configuring all the nodes, we can connect to the Kubernetes cluster from the development machine by copying the Kubernetes configuration file from one of the three master nodes.

First, we install kubectl on the development machine.

```
# Download the kubectl binary  
  
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"  
  
# Download the kubectl checksum file  
  
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"  
  
# Verify the kubectl binary with the checksum file  
  
$ echo "$(cat kubectl.sha256) kubectl" | sha256sum --check  
  
# If verification is successful, install kubectl  
  
$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

On one of the master nodes, we execute the following command to display the Kubernetes credentials:

We copy the contents of the displayed Kubernetes configuration file and move it to `~/kube/config` on the development machine, also changing the IP address from `127.0.0.1` to the IP address of the master node.

3.4 conclusion

In this chapter, we outlined the infrastructure and work environment that support our project. This foundation sets the stage for the next steps, where we will dive into the technologies and tools that will drive our deployment and operations.

Chapter 4: Deployment and Implementation.

4.1 Introduction

In this chapter, we will delve into the deployment and implementation of key components within our infrastructure: the Infinispan Operator, Keycloak Operator, and Nextcloud. We will detail the deployment procedures, configuration settings, and integration steps for each tool. This chapter aims to provide a clear and practical guide to bringing these technologies into our environment, ensuring they are correctly set up and fully operational.

4.2 Infinispan Deployment

4.2.1 Introduction

We will start with deploying the Infinispan Operator, outlining the key steps and configurations needed to integrate it into our Kubernetes cluster. This will set the stage for utilizing Infinispan's distributed caching capabilities effectively.

4.2.2 Deploying the Infinispan Operator

Installing the Infinispan Operator adds Custom Resource Definitions (CRDs) to Kubernetes for managing Infinispan clusters on OpenShift. Users deploy Infinispan clusters by applying Custom Resources (CRs) via the Kubernetes Dashboard or kubectl. The operator then automatically provisions necessary resources, like StatefulSets and Secrets, and configures services based on the CR specifications, including pod numbers and backup locations.

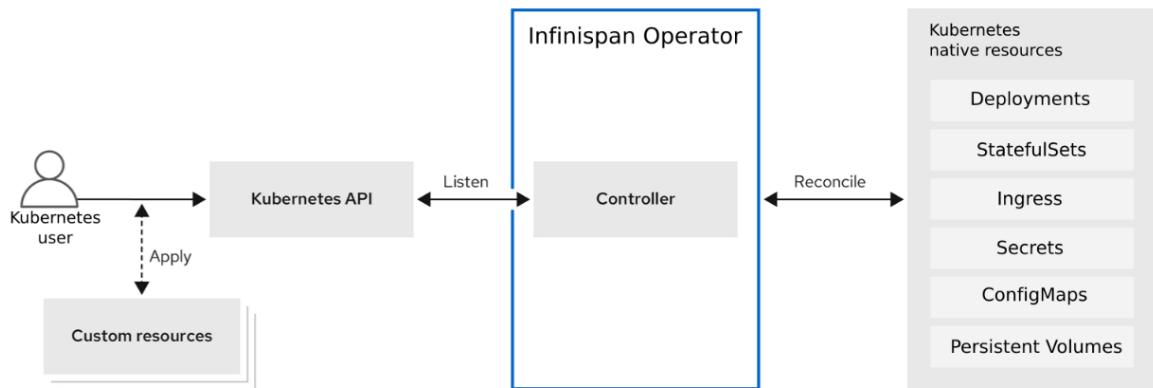


FIGURE 19 OVERVIEW OF THE INFINISPAN OPERATOR WORKFLOW IN KUBERNETES.

To manage Infinispan clusters effectively within Kubernetes, we use the native Infinispan CLI plugin and Custom Resource Definitions (CRDs). The CLI plugin simplifies interaction with Infinispan by adding specific commands to the kubectl tool. Here's how to set it up:

1. INSTALLING THE NATIVE INFINISPAN CLI PLUGIN

Download the Infinispan CLI Distribution:

- Download the native Infinispan CLI distribution from the Infinispan Quarkus releases.

Extract the Distribution:

- Extract the .zip archive of the native Infinispan CLI distribution.

SET UP THE CLI EXECUTABLE:

- COPY THE NATIVE EXECUTABLE OR CREATE A SYMBOLIC LINK TO A FILE NAMED KUBECTL-INFINISPAN

```
cp infinispan-cli kubectl-infinispan
```

ADD TO PATH:

- ADD THE DIRECTORY CONTAINING KUBECTL-INFINISPAN TO YOUR PATH ENVIRONMENT VARIABLE.

VERIFY THE INSTALLATION:

- CHECK THAT THE CLI IS INSTALLED BY RUNNING:

```
worker1@worker1-virtual-machine:~$ kubectl plugin list  
The following compatible plugins are available:  
/usr/local/bin/kubectl-infinispan
```

FIGURE 20 CLI PLUGIN CHECK

VIEW AVAILABLE COMMANDS:

- USE THE FOLLOWING COMMAND TO SEE THE AVAILABLE INFINISPAN COMMANDS:

```
kubectl infinispan --help
```

2. INSTALLING THE OPERATOR LIFECYCLE MANAGER (OLM) AND INFINISPAN CRDs

ENSURE ADMINISTRATOR ACCESS.

- ACCESS TO OPERATORHUB MAY REQUIRE ADMINISTRATOR CREDENTIALS.

INSTALL THE OLM:

- INSTALL THE OPERATOR LIFECYCLE MANAGER (OLM) WITH THE FOLLOWING COMMAND

```
curl -sL https://github.com/operator-framework/operator-lifecycle  
manager/releases/download/v0.28.0/install.sh | bash -s v0.28.0
```

INSTALL INFINISPAN CRDs AND OPERATOR:

```
kubectl create -f https://githubusercontent.com/operator-framework/operator-lifecycle-manager/master/deploy/upstream/quickstart/crds.yaml

kubectl create -f https://githubusercontent.com/operator-framework/operator-lifecycle-manager/master/deploy/upstream/quickstart/olm.yaml
```

- APPLY THE NECESSARY CUSTOM RESOURCE DEFINITIONS (CRDS) FOR OLM:
- DEPLOY THE INFINISPAN OPERATOR FROM OPERATORHUB:

```
kubectl create -f https://operatorhub.io/install/infinispan.yaml
```

VERIFY THE INSTALLATION:

- CONFIRM THAT THE OLM AND INFINISPAN OPERATOR ARE INSTALLED CORRECTLY:

```
kubectl get csv -n operators
```

```
worker1@worker1-virtual-machine:~$ kubectl get csv -n operators
NAME          DISPLAY        VERSION   REPLACES      PHASE
infinispan-operator.v2.4.3  Infinispan Operator  2.4.3    infinispan-operator.v2.4.2  Succeeded
worker1@worker1-virtual-machine:~$
```

FIGURE 21 OPERATOR CHECK PROCESS

3. INFINISPAN DEPLOYMENT MANIFEST FILE

This screenshot captures the manifest file used to deploy Infinispan in a Kubernetes environment. The manifest includes specifications for the Infinispan deployment, service, and necessary configurations such as resource requests and labels.

```

GNU nano 6.2                                infinispan.yaml *
apiVersion: infinitispan.org/v1
kind: Infinispan
metadata:
  name: infinispan-cluster
  namespace: default
spec:
  replicas: 2
  version: 15.0.4
  service:
    type: DataGrid
  expose:
    type: LoadBalancer
  configMapName: "cluster-config"

```

FIGURE 22 INFINISPAN OPERATOR MANIFEST

We create the infinispan.yaml manifest file and then apply it with kubectl apply -f infinispan.yaml. Next, we run kubectl get pods, kubectl get svc, and kubectl get deployment to check the status.

Kubectl apply -f infinispan.yaml

```

worker1@worker1-virtual-machine:~$ kubectl get pods
NAME                               READY   STATUS        RESTARTS   AGE
infinispan-batch-inline-4hw6z      0/1    Completed    0          29h
infinispan-cluster-0              1/1    Running      6 (7m32s ago) 41h
infinispan-cluster-1              1/1    Running      8 (7m46s ago) 41h
infinispan-cluster-config-listener-f47b8cdcf-c7vhn 1/1    Running      7 (6m44s ago) 41h
keycloak-0                        1/1    Running      3 (7m46s ago) 18h

```

FIGURE 23 LIST OF RUNNING PODS

```

worker1@worker1-virtual-machine:~$ kubectl get deployment
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
infinispan-cluster-config-listener  1/1     1           1           7d23h
postgres                 0/1     1           0           35h
worker1@worker1-virtual-machine:~$ 

```

FIGURE 24 KUBERNETES DEPLOYMENTS

```

worker1@worker1-virtual-machine:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
infinispan-cluster   ClusterIP  10.43.235.27 <none>       11222/TCP
4d19h
infinispan-cluster-admin   ClusterIP  None         <none>       11223/TCP
4d19h
infinispan-cluster-external   LoadBalancer  10.43.59.110  192.168.40.137,192.168.40.139,192.168.40.140  11222:30765/TCP
4d19h
infinispan-cluster-ping   ClusterIP  None         <none>       8888/TCP
4d19h

```

FIGURE 25 KUBERNETES SERVICES

4. INFINISPAN CONFIGMAP FILE

The ConfigMap file defines custom configurations and cache templates for the Infinispan cluster, allowing you to fine-tune its behavior and optimize cache management.

```
GNU nano 6.2                               infinispan-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-config
  namespace: default
data:
  infinispan-config.yaml: |
    infinispan:
      cacheContainer:
        caches:
          base-template:
            distributedCacheConfiguration:
              expiration:
                lifespan: "5000"
          extended-template:
            distributedCacheConfiguration:
              configuration: "base-template"
              encoding:
                mediaType: "application/x-protostream"
              expiration:
                lifespan: "10000"
                maxIdle: "1000"
...
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: distributed-cache
  namespace: default
spec:
  clusterName: infinispan-cluster
  name: distcache
  templateName: org.infinispan.DIST_SYNC
...
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
```

FIGURE 26 CONFIGMAP MANIFEST FILE.

```
expiration:
  lifespan: "10000"
  maxIdle: "1000"
---
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: distributed-cache
  namespace: default
spec:
  clusterName: infinispan-cluster
  name: distcache
  templateName: org.infinispan.DIST_SYNC
...
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: replicated-cache
  namespace: default
spec:
  clusterName: infinispan-cluster
  name: replcache
  templateName: org.infinispan.REPL_SYNC
...
apiVersion: infinispan.org/v2alpha1
kind: Batch
metadata:
  name: infinispan-batch-inline
  namespace: default
spec:
  cluster: infinispan-cluster
  config: |
    create cache --template=org.infinispan.DIST_SYNC batchcache
    put --cache=batchcache hello world
    put --cache=batchcache hola mundo
```

FIGURE 27 CONFIGMAP MANIFEST FILE1.

After creating the infinispan-config.yaml, we create the ConfigMap by running the following command

```
Kubectl apply -f infinispan-config.yaml
```

Verify its creation using kubectl get configmap

```
worker1@worker1-virtual-machine:~$ kubectl get configmap
NAME          DATA   AGE
cluster-config    1      4d19h
infinispan-batch-inline 1      4d6h
infinispan-cluster-configuration 4      4d19h
keycloak-startup    1      3d20h
kube-root-ca.crt    1      5d2h
worker1@worker1-virtual-machine:~$
```

FIGURE 28 CHECKING CONFIGMAP DEPLOYMENT

5. LOGGING INTO THE INFINISPAN CONSOLE

To access the Infinispan management console, follow these steps:

1. Obtain the URL: The Infinispan console is typically accessible via the service's external IP or the cluster's domain name. Use kubectl get svc to find the external IP or hostname of your Infinispan service.
2. Access the Console: Open a web browser and navigate to the URL in the format `http://<external-ip>:<port>` or `https://<hostname>:<port>`.
3. Login Credentials: Use the username and password that were set during the Infinispan configuration. If credentials were stored in a Kubernetes secret, you might need to retrieve them using `kubectl get secret`.
4. Explore the Console: Once logged in, you can manage caches, view metrics, and configure additional settings directly from the console.

```
infinispan-operator:v2.1.2   infinispan-operator   v2.1.3   infinispan-operator:v2.1.2
worker1@worker1-virtual-machine:~$ kubectl get secrets
NAME          TYPE      DATA   AGE
infinispan-cluster-generated-operator-secret  Opaque     4      4d19h
infinispan-cluster-generated-secret        Opaque     1      4d19h
infinispan-cluster-infinispan-security    Opaque     1      4d19h
infinispan-cluster-infinispan-security    Opaque     1      4d19h
```

FIGURE 29 CHECKING SECRETS.

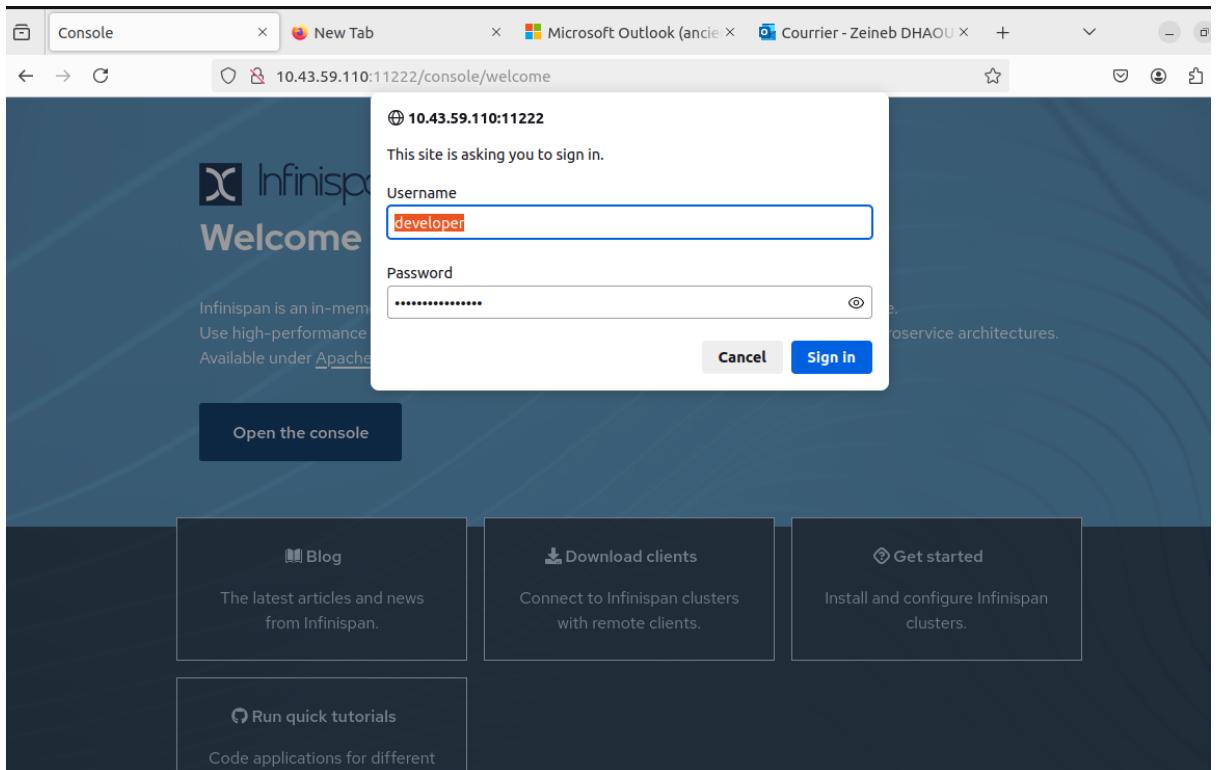


FIGURE 30 INFINISPAN WEB INTERFACE.

A screenshot of the Infinispan Admin Console. The URL is 10.43.59.110:11222/console. The left sidebar shows "Data Container", "Global Statistics", "Cluster Membership", "Access Management", and "Connected Clients". The main panel is titled "Data container" and shows a table of caches. The table has columns: Name, Type, Health, and Features. Five caches are listed: batchcache (Distributed, Healthy), distcache (Distributed, Healthy), memcachedCache (Replicated, Healthy), replcache (Replicated, Healthy), and respCache (Distributed, Healthy).

Name	Type	Health	Features
batchcache	Distributed	Healthy	None
distcache	Distributed	Healthy	None
memcachedCache	Replicated	Healthy	None
replcache	Replicated	Healthy	None
respCache	Distributed	Healthy	None

FIGURE 31 INFINISPAN ADMIN CONSOLE.

4.3 Deploying Postgres

4.3.1 Deploying PostgreSQL in a Kubernetes Environment

1. Preparing the environment

Before you deploy PostgreSQL, you need to make sure that your Kubernetes cluster is up and running. You can check the status of the cluster and nodes by running the following command:

```
kubectl get nodes
```

```
root@masternode2:/home/masternode2# kubectl get nodes
NAME      STATUS    ROLES     AGE      VERSION
masternode1 Ready     control-plane,master   44h      v1.30.4+k3s1
masternode2 Ready     control-plane,master   45h      v1.30.4+k3s1
workernode1 Ready     <none>    44h      v1.30.4+k3s1
workernode2 Ready     <none>    44h      v1.30.4+k3s1
```

FIGURE 32 VERIFYING CLUSTER NODES.

2. Creating a ConfigMap for PostgreSQL Configuration

Create a ConfigMap to manage PostgreSQL configuration settings. This can be used to define environment variables or configuration files required by PostgreSQL. For example, you can include settings for initializing the database. Apply the ConfigMap using the following:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-secret
  labels:
    app: postgres
data:
  POSTGRES_DB: keycloak_db
  POSTGRES_USER: keycloak_user
  POSTGRES_PASSWORD: keycloak123
```

FIGURE 33 POSTGRESQL CONFIGMAP SETUP.

```
kubectl apply -f postgres-configmap.yaml
```

This command creates a ConfigMap based on the parameters defined in the YAML file, which can be referenced by your PostgreSQL deployment.

3. Creating a PostgreSQL Deployment

Define the Deployment resource for PostgreSQL. This ensures that PostgreSQL runs as a containerized application within a pod. Apply the deployment configuration file by using the following:

```
kubectl apply -f postgres-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: 'postgres:16'
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5432
          envFrom:
            - configMapRef:
                name: postgres-secret
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgresdata
      volumes:
        - name: postgresdata
          persistentVolumeClaim:
            claimName: postgres-volume-claim
```

FIGURE 34 POSTGRESQL DEPLOYMENT MANIFEST FILE.

This command tells Kubernetes to create the PostgreSQL deployment according to the settings set in the YAML file.

4. Check PostgreSQL space

After applying the deployment, it is essential to verify that the PostgreSQL pod is running correctly. You can check the status of the pod using the following:

```
kubectl get pods -l app=postgres
```

This command lists all the pods that are running in the cluster, indicating the status of each pod (for example, Running, Pending, or Error). This is to ensure that the PostgreSQL pod has been successfully created and is up and running.

```
root@masternode2:/home/masternode2/cluster# kubectl get pods -l app=postgres
NAME           READY   STATUS    RESTARTS   AGE
postgres-7d554995f8-s4njf   1/1     Running   1 (4h52m ago)   46h
root@masternode2:/home/masternode2/cluster#
```

FIGURE 35 CHECKING POSTGRESQL POD STATUS

5. Creating a PersistentVolume (PV)

Before you create a PersistentVolumeClaim (PVC), you need a PersistentVolume (PV), which is the actual storage resource allocated to your database. The PV provides physical storage and must be created using the following command:

```
kubectl apply -f postgres-pv.yaml
```

This command provisions the physical storage that the PostgreSQL instance will use to store its data.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-volume
  labels:
    type: local
    app: postgres
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /data/postgresql
```

FIGURE 36 POSTGRESQL PERSISTENT STORAGE CONFIGURATION.

6. Creating a PersistentVolumeClaim (PVC)

A PVC card is used to request storage for your PostgreSQL instance. To create a permanent virtual circuit, run the following command:

```
kubectl apply -f postgres-pvc.yaml
```

PVC dynamically binds to a PV, which ensures that PostgreSQL has persistent storage for its data.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-volume-claim
  labels:
    app: postgres
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

FIGURE 37 POSTGRESQL PVC CONFIGURATION

7. Check the condition of the PVC

Ensure that the PVC is properly bonded to a PV by performing:

```
Kubectl get pvc
```

This command lists the permanent virtual circuits and indicates whether they are linked, which means that the PostgreSQL instance has access to persistent storage.

```
root@masternode2:/home/masternode2/cluster# kubectl get pvc
NAME           STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
postgres-volume-claim   Bound    postgres-volume   10Gi       RWX          manual        <unset>           47h
root@masternode2:/home/masternode2/cluster#
```

FIGURE 38 CHECKING PVC POD STATUS

4.4 Keycloak deployment

4.4.1 Keycloak operator deployment

- Make sure that OLM is installed in your environment. For more details, see the section on installing OLM.
- Verify that you have cluster administrator permissions or an equivalent level of permissions granted by an administrator.
- Go to the Operator Hub, find the Keycloak operator, and click the Install button. Follow the instructions displayed to complete the installation.

```
worker1@worker1-virtual-machine: $ kubectl get ols
NAME                                     AGE
operator.operators.coreos.com/infinispan.operators   42h
operator.operators.coreos.com/keycloak-operator.my-keycloak-operator 18h
```

FIGURE 39 CHECKING KUBERNETES OLM.

- Deploy the database.

Checking Keycloak's connection to PostgreSQL:

1. PostgreSQL and Keycloak connection

PostgreSQL and Keycloak are successfully connected, allowing Keycloak to store and manage its data in the PostgreSQL database.

`kubectl get svc`

```
root@masternode2:/home/masternode2/cluster# kubectl get svc postgres
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
postgres  LoadBalancer  10.43.241.20  <pending>      5432:30405/TCP  47h
```

FIGURE 40 CHECKING POSTGRESQL SERVICES

2. View Keycloak tables in PostgreSQL

- To view Keycloak tables in PostgreSQL, follow these steps. First, navigate to the PostgreSQL container that runs in your Kubernetes cluster. Once inside the container, connect to PostgreSQL and run the query to list the tables.

```
# Access the PostgreSQL pod
kubectl exec -it <nom-pod-postgres> -- /bin/bash

# Connect to PostgreSQL
psql -U <postgres-username> -d <keybase>

# See the list of Keycloak tables
\German
```

List of relations				
Schema	Name	Type	Owner	
public	admin_event_entity	table	keycloak_user	
public	associated_policy	table	keycloak_user	
public	authentication_execution	table	keycloak_user	
public	authentication_flow	table	keycloak_user	
public	authenticator_config	table	keycloak_user	
public	authenticator_config_entry	table	keycloak_user	
public	broker_link	table	keycloak_user	
public	client	table	keycloak_user	
public	client_attributes	table	keycloak_user	
public	client_auth_flow_bindings	table	keycloak_user	
public	client_initial_access	table	keycloak_user	
public	client_node_registrations	table	keycloak_user	
public	client_scope	table	keycloak_user	
public	client_scope_attributes	table	keycloak_user	
public	client_scope_client	table	keycloak_user	
public	client_scope_role_mapping	table	keycloak_user	
public	client_session	table	keycloak_user	
public	client_session_auth_status	table	keycloak_user	
public	client_session_note	table	keycloak_user	

FIGURE 41 VIEWING KEYCLOAK TABLES IN POSTGRESQL

- Configure the hostname. (For development purposes, this guide will use test.keycloak.org.)
- Configure the TLS certificate and key. For development purposes, you can use the following command to generate a self-signed certificate:

```
openssl req -subj '/CN=test.keycloak.org/O=Test Keycloak./C=US' -newkey rsa :2048 -nodes -keyout key.pem -x509 -days 365 -out certificate.pem
```

You must install it in the cluster namespace as a secret by entering this command:

```
kubectl create secret tls example-tls-secret \
--cert=certificate.pem \
--key=clé.pem \
-n my-keycloak-operator
```

Run kubectl get secrets -n keycloak to verify that the keycloak-tls-secret has been created successfully.

```
worker1@worker1-virtual-machine:~$ kubectl get secrets -n my-keycloak-operator
NAME                      TYPE           DATA   AGE
example-keycloak-initial-admin  kubernetes.io/basic-auth  2      99m
example-tls-secret          kubernetes.io/tls       2      50m
keycloak-db-secret          Opaque          2      45m
keycloak-tls-secret          kubernetes.io/tls       2      100m
worker1@worker1-virtual-machine:~$ █
```

FIGURE 42 KEYCLOAK SECRETS

4.4.2 Deploying Keycloak

To deploy Keycloak, you must create a custom resource (CR) based on the Keycloak custom resource definition (CRD).

Consider storing the database credentials in a separate secret. Enter the following commands:

```
kubectl create secret generic keycloak-db-secret \
--from-literal=nomduutilisateur=[your_database_username] \
--from-literal=password=[your_database_password]
```

```

worker1@worker1-virtual-machine:~$ kubectl create secret generic keycloak-db-secret \
--from-literal=username=testuser \
--from-literal=password=testpassword \
-n my-keycloak-operator
secret/keycloak-db-secret created
worker1@worker1-virtual-machine:~$
```

FIGURE 43 CREATING DATABASE SECRETS FOR KEYCLOAK.

Create a keycloak.yaml YAML file

```

GNU nano 6.2                                     keycloak.yaml
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: keycloak
  namespace: my-keycloak-operator
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: example-tls-secret
    hostname:
      hostname: test.keycloak.org
    proxy:
      headers: xforwarded # double check your reverse proxy sets and overwrites the X-Forwarded-* headers
```

FIGURE 44 KEYCLOAK DEPLOYMENT MANIFEST FILE

Kubectl apply -f keycloak.yaml

```

error: the server doesn't have a resource type "statefulset"
worker1@worker1-virtual-machine:~$ kubectl get statefulset
NAME          READY   AGE
infinispan-cluster   1/2    4d19h
keycloak        1/1    3d20h
keycloak-postgresql 1/1    3d20h
worker1@worker1-virtual-machine:~$
```

FIGURE 45 KUBERNETES STATEFULSET SCREENSHOT.

```

worker1@worker1-virtual-machine:~$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
infinispan-batch-inline-4hw6z   0/1    Completed  0          29h
infinispan-cluster-0           1/1    Running   6 (7m32s ago)  41h
infinispan-cluster-1           1/1    Running   8 (7m46s ago)  41h
infinispan-cluster-config-listener-f47b8cdcf-c7vhn 1/1    Running   7 (6m44s ago)  41h
keycloak-0                     1/1    Running   3 (7m46s ago)  18h
keycloak-1                     0/1    Pending   0          18h
```

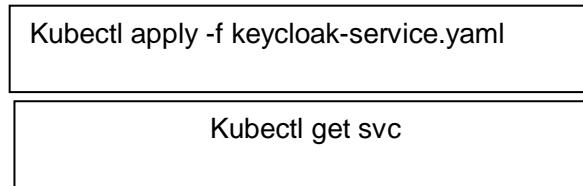
FIGURE 46 KUBERNETES PODS SCREENSHOT

4.4.3 Accessing the Keycloak deployment

Create a Kubernetes Service of type LoadBalancer to expose Keycloak:

```
GNU nano 6.2
apiVersion: v1
kind: Service
metadata:
  name: keycloak
  namespace: keycloak
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
    - port: 443
      targetPort: 8443
  selector:
    app: keycloak
```

FIGURE 47 KEYCLOAK SERVICE MANIFEST.



NAME	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
infinispan-cluster	4d19h	ClusterIP	10.43.235.27	<none>	11222/TCP
infinispan-cluster-admin	4d19h	ClusterIP	None	<none>	11223/TCP
infinispan-cluster-external	4d19h	LoadBalancer	10.43.59.110	192.168.40.137,192.168.40.139,192.168.40.140	11222:30765/TCP
infinispan-cluster-ping	4d19h	ClusterIP	None	<none>	8888/TCP
keycloak	599/TCP 3d21h	LoadBalancer	10.43.116.195	192.168.40.137,192.168.40.139,192.168.40.140	8080:32629/TCP,8443:30
keycloak-headless	3d20h	ClusterIP	None	<none>	80/TCP
keycloak-http	CP 3d20h	ClusterIP	10.43.24.71	<none>	80/TCP,8443/TCP,9990/T
keycloak-postgresql	3d20h	ClusterIP	10.43.71.41	<none>	5432/TCP
keycloak-postgresql-headless	3d20h	ClusterIP	None	<none>	5432/TCP
kubernetes	5d2h	ClusterIP	10.43.0.1	<none>	443/TCP

FIGURE 48 KUBERNETES SERVICES OVERVIEW.

4.4.4 Starting keycloak

Discovering the Keycloak admin and account consoles In this section, we will take a look at the Keycloak admin and account consoles. The admin console is an extensive console

that allows you to configure and manage Keycloak. The account console, on the other hand, is there to allow your end users to manage their accounts.

Getting started with the Keycloak admin console

The Keycloak admin console provides an extensive and friendly interface for administrators and developers to configure and manage Keycloak. To access the admin console, open <http://localhost:8080/admin> in a browser. You will be redirected to the Keycloak login page, where you can log in with the admin username and password you created in the previous section while installing Keycloak. Once you have logged in, you will see the configuration for the master realm in Keycloak, as shown in the following screenshot:

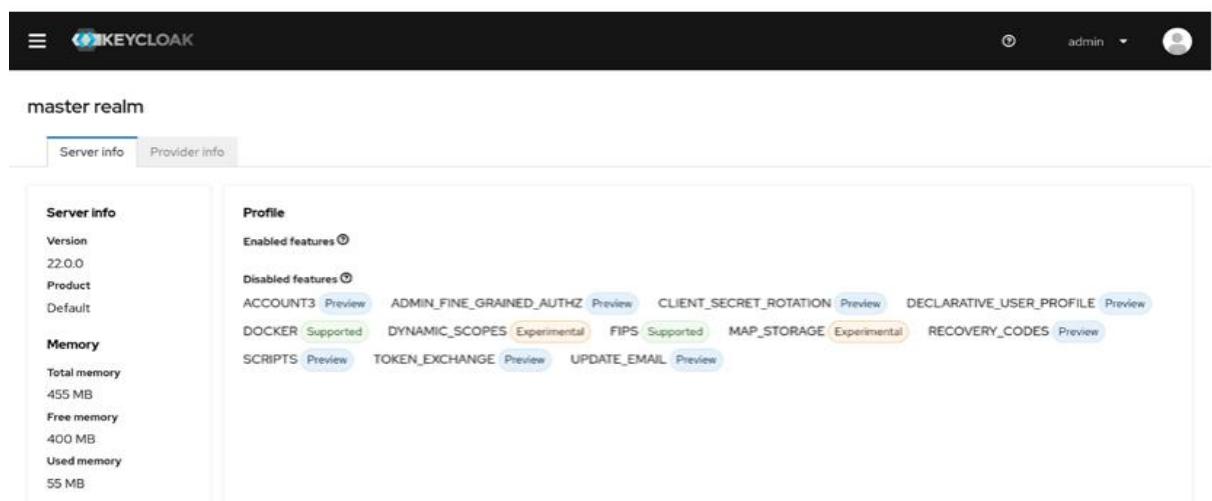
The screenshot shows the Keycloak Admin Console interface. At the top, there's a dark header bar with the Keycloak logo and a user dropdown set to 'admin'. Below the header, the title 'master realm' is displayed. Underneath the title, there are two tabs: 'Server info' (which is selected) and 'Provider info'. The main content area is divided into two columns. The left column, titled 'Server info', contains sections for 'Version' (22.0.0), 'Product' (Default), and 'Memory' (Total memory: 455 MB, Free memory: 400 MB, Used memory: 55 MB). The right column, titled 'Profile', shows 'Enabled features' including ACCOUNT3, ADMIN_FINE_GRAINED_AUTHZ, CLIENT_SECRET_ROTATION, DECLARATIVE_USER_PROFILE, DOCKER (Supported), DYNAMIC_SCOPES (Experimental), FIPS (Supported), MAP_STORAGE (Experimental), RECOVERY_CODES, and SCRIPTS. It also lists 'Disabled features' like TOKEN_EXCHANGE and UPDATE_EMAIL.

FIGURE 49 THE KEYCLOAK ADMIN CONSOLE

Creating and configuring a realm

To create a realm in Keycloak, follow these steps:

1. Expand the Menu: Click on the menu icon in the top-left corner next to the Keycloak logo to expand the navigation menu.
2. Access the Realm Selector: Click on the realm selector to view the list of existing realms.
3. Create a New Realm: In the realm selector, click on the "Create Realm" button.

This will allow you to set up a new realm, which is fully isolated from other realms with its own configuration, users, and applications.

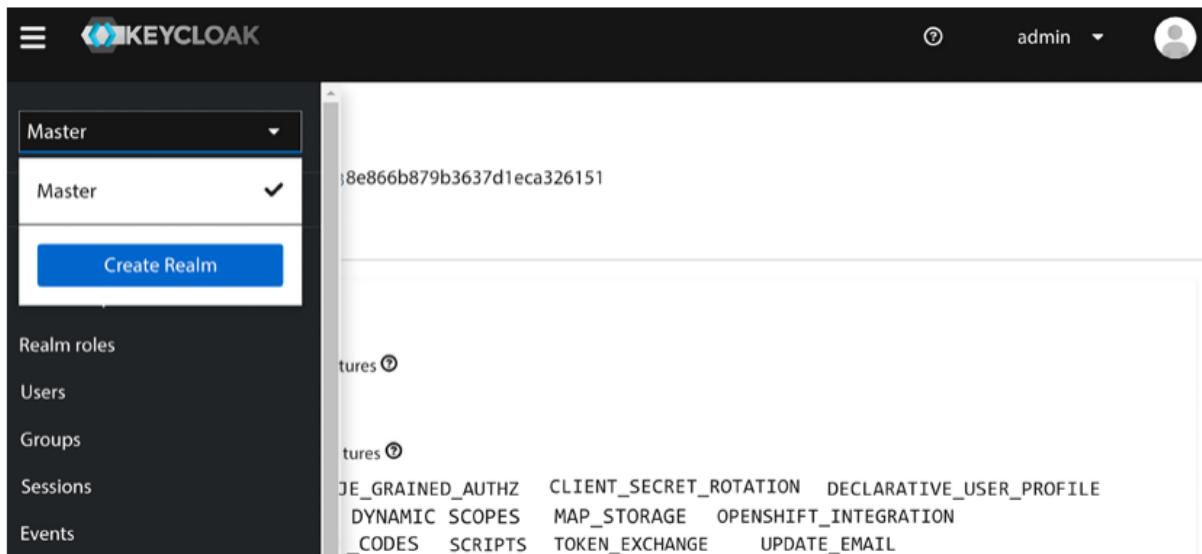


FIGURE 50 : REALM SELECTOR.

On the next page, enter a name for your new realm. Since the name will be part of URLs, it's best to avoid special characters that require escaping (e.g., spaces). After creating the realm, you can assign a more user-friendly display name. For instance, you might use "myrealm" as the name and "My Realm" as the display name.

Creating a user

To create a user in the realm:

1. Navigate to the **Users** section in the left-hand menu, then click on the **Create new user** button.
2. Fill in the fields for the username, email, first name, and last name with memorable and relevant information.
3. If you are sure the email address is valid, you can select the **Email Verified** option.
4. In the **Required User Actions** section, you can mandate certain actions for the user upon their next login, such as reviewing their profile or verifying their email.
5. After completing the form, ensure you click on **Create** to save the new user.

The screenshot shows the 'Create user' form in a Keycloak interface. At the top, there's a breadcrumb navigation: 'Users > Create user'. The main title is 'Create user'. Below it, there's a section titled 'Required user actions' with a dropdown menu set to 'Select action'. The form fields are as follows:

- Username ***: keycloak
- Email**: keycloak@keycloak.org
- Email verified**: No (with a toggle switch)
- First name**: Ola
- Last name**: Nordmann
- Groups**: Join Groups (with a button)

At the bottom of the form are two buttons: 'Create' (in blue) and 'Cancel'.

FIGURE 51 THE ADD USER FORM.

A user has a few standard built-in attributes, such as First Name, but you can also add custom attributes through the **Attributes** tab.

Before the user can log in, you'll need to set an initial temporary password. To do this, navigate to the **Credentials** tab. In this tab, click on the **Set Password** button and follow the instructions to set a new password for the user.

If the **Temporary** option is enabled, the user will be required to change their password upon their first login. This is particularly useful when an administrator creates the user, as it ensures the user sets a secure password.

Creating a Group

Next, let's create a group and add the user you previously created to it.

1. From the left-hand menu, click on **Groups**, and then click on the **Create group** button.
2. Enter a name for the group, such as mygroup, and then click on **Create**.

Once you've created the group, you'll see mygroup in the group list. By clicking on it, you can edit the group settings. For instance:

- You can add attributes to the group, which will be inherited by all users in the group. For example, if you have a group for all employees in an office, you could add the office address to the group, and it would be automatically applied to all members.
- You can also assign roles to the group, which will be inherited by all members of the group.

To add the user to the group:

1. Go back to the **Users** page and select the user you created earlier.
2. Click on the **Groups** tab.
3. Click **Join Group**, select the group you created (mygroup), and click **Join** to add the user to the group.

Creating a Global Role

To create a global role:

1. Click on **Realm roles** in the left-hand menu, and then click on **Create role**.
2. Enter a role name, such as myrole, and add a description if desired.

Roles in Keycloak can be composite, meaning they can include other roles. A user assigned to a composite role will inherit all the roles within it. While powerful, composite roles should be used cautiously to avoid complexity and performance overhead, especially with many nested roles.

To assign the user to the role:

1. Go back to the **Users** page and select the user you created earlier.
2. Click on the **Role mapping** tab.
3. Click **Assign role**, select the role you created (myrole), and click **Assign** to add the user to the role.

4.5 Deploying Nextcloud and Integrating with Keycloak for SSO

4.5.1 Deploying Nextcloud

1. **Create a PersistentVolume for Nextcloud Storage**

First, create the PersistentVolume (PV) for Nextcloud storage, which will store user uploads and application data. Apply the following command to provision the required storage:

```
kubectl apply -f nextcloud-pv.yaml
```

```
# nextcloud-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nextcloud-pv
  labels:
    type: local
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  hostPath:
    path: /mnt/data
```

FIGURE 53 NEXTCLOUD PV MANIFEST FILE

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	VOLUMEATTRIBUTESCLASS
nextcloud-pv	2Gi	RWO	Retain	Available		manual	<unset>

FIGURE 52 NEXTCLOUD PVC CHECK

2. Create a PersistentVolumeClaim for Nextcloud

After setting up the PV, create a PersistentVolumeClaim (PVC) to allow Nextcloud to access the allocated storage. This ensures that the Nextcloud pod has persistent storage. Apply the PVC configuration using:

```
kubectl apply -f nextcloud-pvc.yaml
```

```
# nextcloud-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nextcloud-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
  selector:
    matchLabels:
      type: local
```

FIGURE 54 NEXTCLOUD PVC MANIFEST FILE

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	VOLUMEATTRIBUTESCLASS	AGE
nextcloud-pvc	Bound	nextcloud-pv	2Gi	RWO	manual	<unset>	4s

FIGURE 55 NEXTCLOUD PVC STATUS

3. Deploy Nextcloud in Kubernetes

With storage ready, deploy Nextcloud by applying the deployment YAML. This file configures how the Nextcloud pod is run inside the Kubernetes environment:

```
kubectl apply -f nextcloud-deployment.yaml
```

```
metadata:
  name: nextcloud
  namespace: default
  labels:
    app: nextcloud
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nextcloud
  template:
    metadata:
      labels:
        app: nextcloud
    spec:
      containers:
        - name: nextcloud
          image: nextcloud:latest
          ports:
            - containerPort: 80
          resources:
            requests:
              memory: "512Mi"
              cpu: "250m"
            limits:
              memory: "1Gi"
              cpu: "500m"
          volumeMounts:
            - mountPath: /var/www/html
              name: nextcloud-storage
      volumes:
        - name: nextcloud-storage
          persistentVolumeClaim:
            claimName: nextcloud-pvc
```

FIGURE 56 NEXTCLOUD DEPLOYMENT MANIFEST FILE

4. Verify the Nextcloud Pod

Once deployed, verify that the Nextcloud pod is running by checking its status using:

```
kubectl get pods
```

```
root@masternode2:/home/masternode2/cluster# kubectl get pod -l app=nextcloud
NAME                               READY   STATUS        RESTARTS   AGE
nextcloud-569b58d8b8-p68v7     0/1    ContainerCreating   0          105s
root@masternode2:/home/masternode2/cluster# kubectl get pod -l app=nextcloud
NAME                               READY   STATUS        RESTARTS   AGE
nextcloud-569b58d8b8-p68v7     1/1    Running      0          3m12s
root@masternode2:/home/masternode2/cluster#
```

FIGURE 57 NEXTCLOUD PODS STATUS

5. Access Nextcloud for Initial Setup

After the Nextcloud pod is running, open your browser and navigate to the external IP or URL for your Nextcloud instance. This will take you to the initial setup page, where you will create an admin account by providing a username and password for the Nextcloud administrator

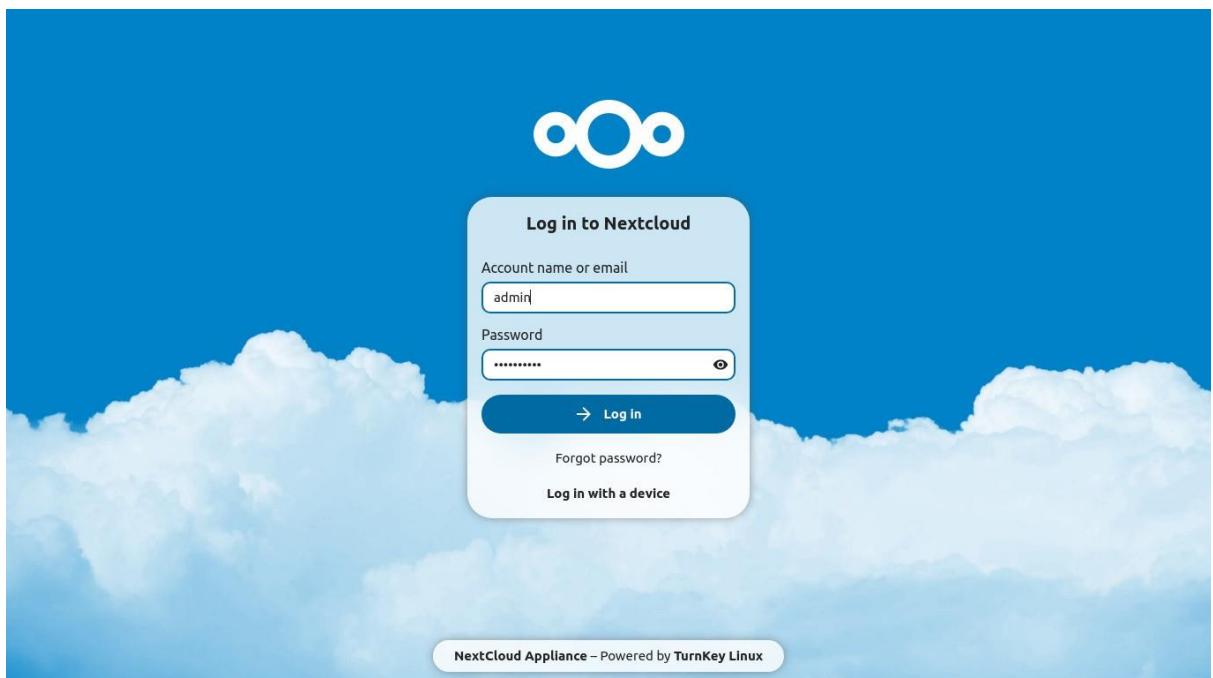


FIGURE 58 NEXTCLOUD WELCOME PAGE.

6. Create a Realm in Keycloak

Log in to the Keycloak admin console and create a new realm dedicated to Nextcloud. This realm will isolate users and clients specifically for managing Nextcloud authentication. Name the realm appropriately (e.g., NextcloudRealm).

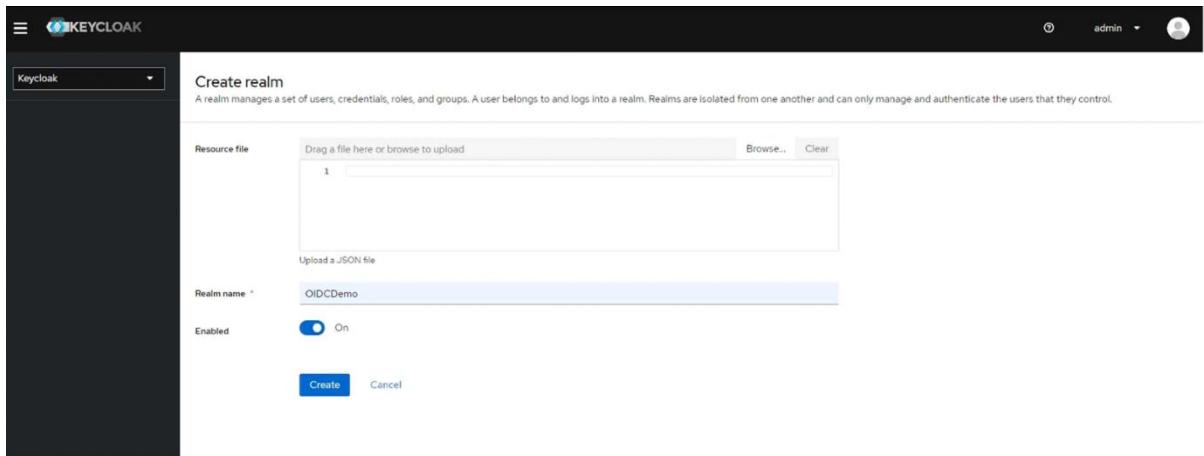


FIGURE 59 CREATE REALM

7. Create a Client for Nextcloud in Keycloak

After creating the realm, set up a client that allows Keycloak to handle authentication for Nextcloud. In the Keycloak realm, go to **Clients** and create a new client with the following settings:

- **Client ID:** nextcloud
- **Protocol:** openid-connect
- **Access Type:** confidential
- **Redirect URI:** <http://nextcloud.your-domain.com/>

FIGURE 60 CREATE CLIENT

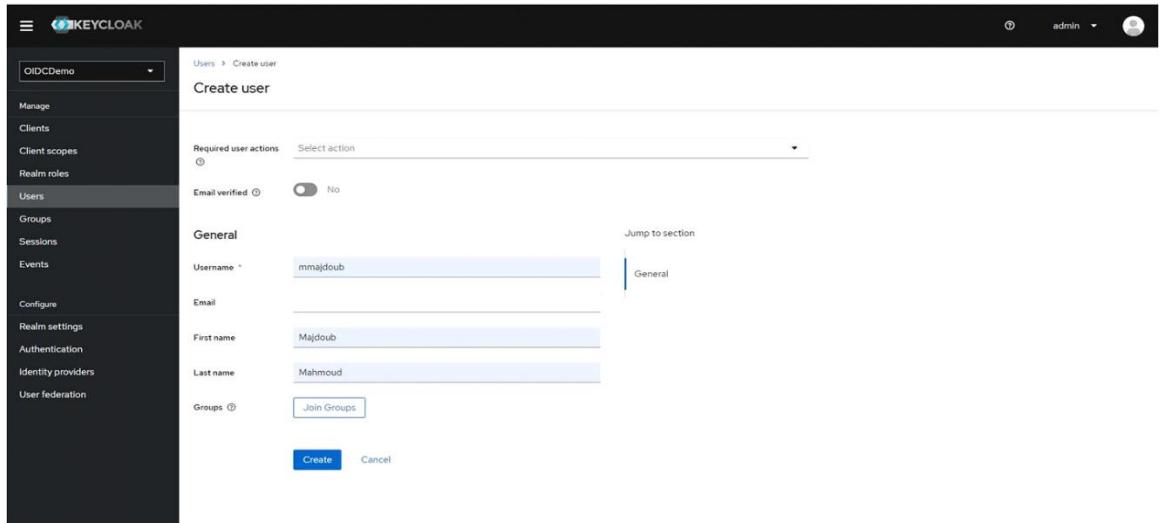


FIGURE 61 CREATE USER

8. Configure OpenID Connect in Nextcloud

After installing the OpenID Connect Login app in Nextcloud, you'll need to configure it to integrate with Keycloak for Single Sign-On (SSO). Follow these steps to set up the integration:

- Go to **Settings** in Nextcloud.
- Navigate to **Administration** and select **OpenID Connect Login**.
- Enter the following details:
 - **Provider URL:** <http://keycloak.your-domain.com/auth/realm/YourRealm>
 - **Client ID:** nextcloud
 - **Client Secret:** (from Keycloak client configuration)
 - **Scopes:** openid profile email
 - **Redirect URI:** <http://nextcloud.your-domain.com/index.php/apps/openidconnect/redirect>
- Save the configuration to finalize the integration.

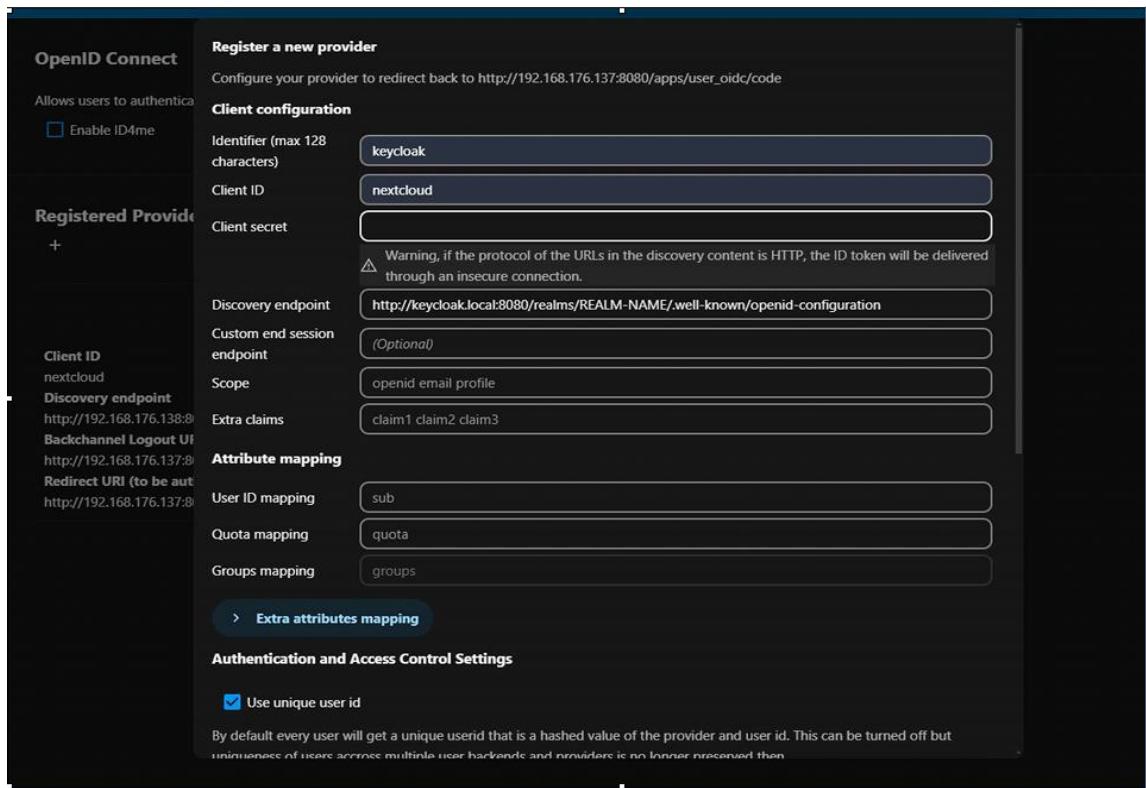


FIGURE 62 OPENID CONFIGURATION

9. Test SSO Integration with Keycloak

After setting up the OpenID Connect app, navigate to the Nextcloud login page. You should now see an option to log in using Keycloak. To test the integration:

1. Click **Login with Keycloak**.
2. You will be redirected to the Keycloak login page.
3. Enter your Keycloak credentials.

- Once authenticated, you will be returned to Nextcloud, where you'll be logged in automatically.

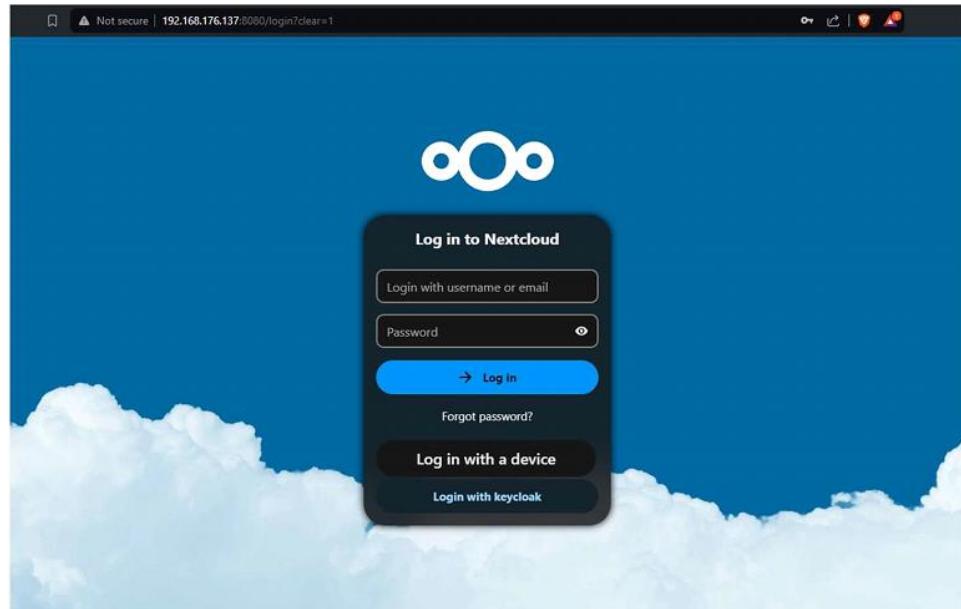


FIGURE 63 NEXTCLOUD LOGIN PAGE

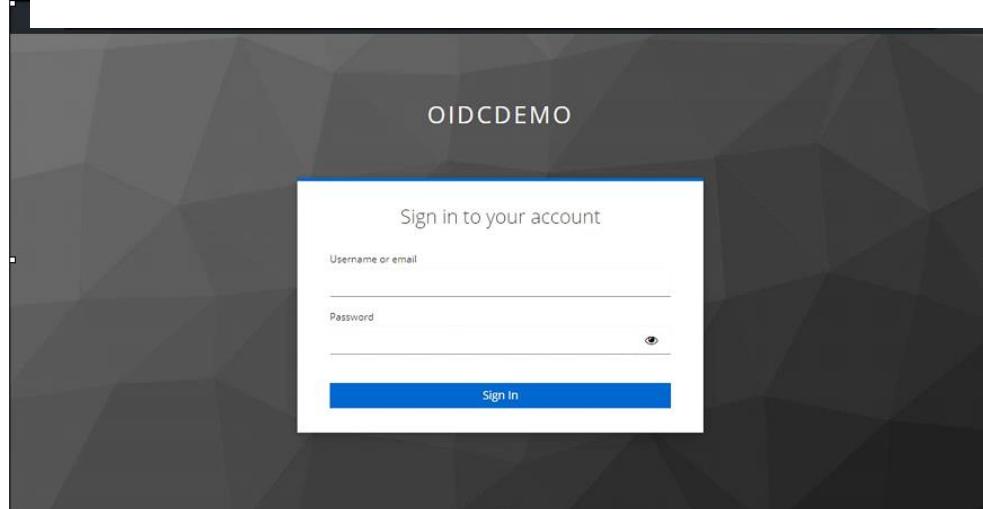


FIGURE 64 LOGIN TO NEXTCLOUD WITH KEYCLOAK

10. Enter Nextcloud Using Keycloak

Finally, you can now access Nextcloud using Keycloak for Single Sign-On (SSO). After successfully authenticating through Keycloak, you will be redirected back to Nextcloud and logged in without needing to manually enter credentials again.

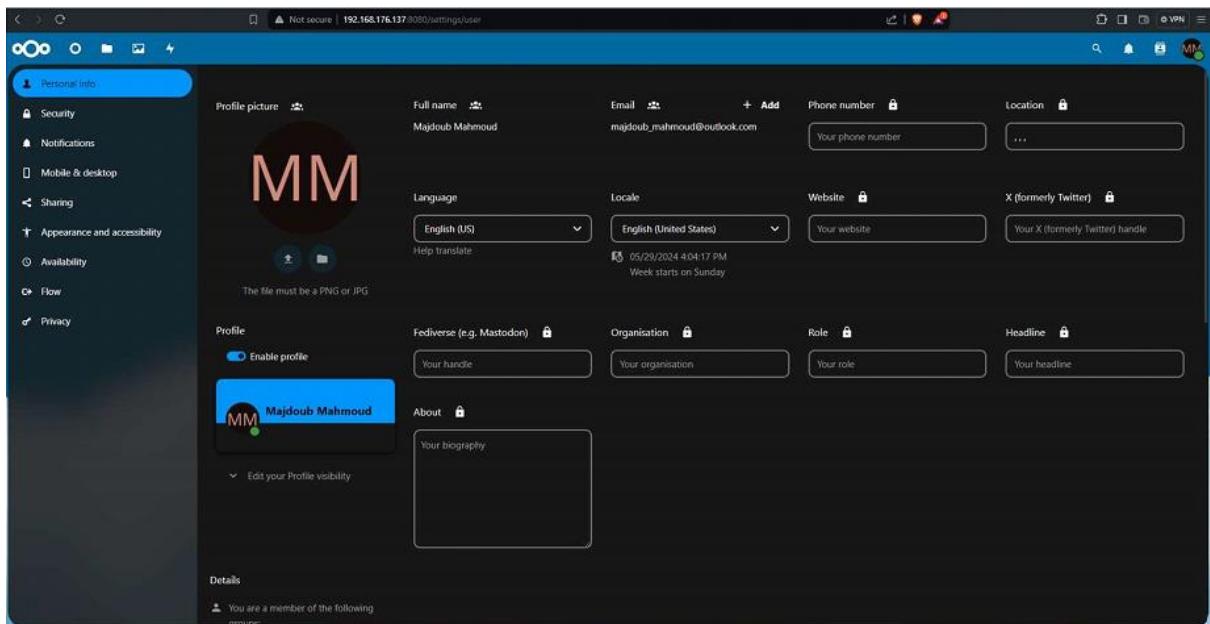


FIGURE 65 WELCOME PAGE TO NEXTCLOUD.

4.6Conclusion

This chapter demonstrated how to integrate Keycloak for identity management, Infinispan for caching, a database for persistence, and Nextcloud for file sharing. Together, these components provide a secure, scalable, and efficient solution for managing user access and collaboration.

General Conclusion

In this project, we have successfully leveraged the power of modern DevOps practices and IAM solutions to build a robust, high-availability infrastructure. By deploying Keycloak for identity management, Infinispan for distributed caching, and PostgreSQL for data storage, we have created a resilient and scalable system that meets contemporary requirements for security and performance.

The integration of these technologies within a Kubernetes environment underscores the effectiveness of DevOps in managing complex deployments and ensuring high availability. Keycloak's configuration for fault tolerance, coupled with Infinispan's efficient caching and PostgreSQL's reliable data management, exemplifies how modern tools and practices can address the challenges of securing and scaling applications.

This project not only demonstrates the benefits of adopting advanced DevOps and IAM solutions but also sets a benchmark for future implementations, paving the way for continued innovation and improvement in managing and securing digital assets.