MedTech
Mediterranean
Institute of Technology

SOFTWARE ENGINEERING

FINAL REPORT

Automated Tests for a Cloud File Storage System

BY

Skander Lahbaiel

ACADEMIC SUPERVISOR

Amina Mseddi

INSTITUTION SUPERVISOR

Aymen Frikha

LiJo Technologies

Tunis, 2023-2024

APPROVAL

APPROVED BY

ACADEMIC SUPERVISOR

Name	Signature	Date
Amina Mseddi		

COMPANY SUPERVISOR

Name	Signature	Date
Aymen Frikha		

ACADEMIC EVALUATOR

Name	Signature	Date

DECLARATION

I certify that I am the author of this project and that any assistance I received in its preparation is fully acknowledged and disclosed in this project.

I have also cited any source from which I used data, ideas, or words, either quoted or paraphrased. Further, this report meets all the rules of quotation and referencing in use at MedTech, as well as adhering to the fraud policies listed in the MedTech honor code.

No portion of the work referred to in this study has been submitted in support of an application for another degree or qualification to this or any other university or institution of learning.

I also certify that this final version of my capstone project report includes the corrections and comments mentioned by the jury members and it is submitted online on Moodle.

Student Name Signature

Date

WORK TERM RELEASE

I hereby state and verify with my signature that I have reviewed this report. I hereby affirm that the report contains:

☐ no confidential data/information, and I authorize it to be released.

☐ confidential data/information, and I do not authorize it to be released.

COMPANY SUPERVISOR

Name	Signature	Date

ABSTRACT

The study explores the deployment of NextCloud on bare-metal cloud infrastructure to research functional and security testing and then hardening, and to be integrated into the CI/CD pipeline. The Design Science Research paradigm was applied and facilitated the development of the work. The implementation of the deployment, testing, and hardening of security was done in an iterative manner. Its main objective was to automate the testing and hardening process and integrate them into a CI pipeline to assure all new increments of the project.

The major key findings of the implemented strategy were increased confidence in the core functionalities of the application. Additionally, enhanced quality and security metrics and the implementation of an integrated automated test through the CI/CD pipeline in NextCloud. The initial audits following the CIS benchmarks indicated that there were numerous vulnerabilities and issues of non-compliance. The scores on compliance significantly improved post-hardening measures, which highlighted the efficiency of the automated testing and hardening in a CI/CD pipeline. This proved automation is a must-have in securing and maintaining the reliability of cloud-native applications.

In this case, some challenges were encountered when dealing with the setup complexity of the tests, the increasing pipeline execution time, and the automation of simulating user interactions with the system to implement functional tests. These were coped with by making heavy use of documentation, configuration management tools, and the restructuring of workflows so testing phases could be introduced without affecting the flow of other processes.

This study highlights the importance of automation in maintaining secure, efficient cloud-native applications. The findings provide insights for both researchers and industry practitioners, emphasizing continuous security monitoring and early issue detection to deliver secure, resilient applications.

Keywords: CI/CD pipeline, Automated testing, Security hardening, CIS benchmarks, Kubernetes, Functional testing, Infrastructure testing, Automated documentation

DEDICATIONS

I dedicate this work to **my father** for his invaluable advice, **my mother** for her unwavering support, and both for their enduring love and sacrifices.

To **my brothers** for their care and constant encouragement. To my **closest friends** for their support and companionship. And **to professors and university staff** who have guided me through these five years of engineering.

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to all those who have supported me throughout this journey. This project is the culmination of five years of dedication, growth, and learning, and I am deeply thankful to everyone who played a part in it.

First and foremost, I am profoundly grateful to my academic supervisor, **Dr. Amina Mseddi**. Her unwavering support, insightful guidance, and constant encouragement have been pivotal throughout this project and my entire engineering education. Her wisdom and advice have been invaluable, and I am truly thankful for her mentorship.

I also wish to express my sincere appreciation to my professional supervisor, **Dr. Aymen Frikha**. His expert advice, continuous support, and motivating words were essential in navigating the challenges of this research. His commitment to my success and his insightful feedback have been greatly appreciated.

A special thanks to my coworker, **Selim Masmoudi**, whose assistance and collaboration were instrumental in the completion of this project. His readiness to help and his valuable contributions made a significant difference.

I am also grateful to my university, my professors, and all the staff members who have provided a nurturing and stimulating academic environment. The knowledge and experiences gained here have been transformative and deeply enriching.

To my friends, thank you for your unwavering support and companionship. Your encouragement and understanding have been a constant source of strength.

Lastly, my deepest appreciation goes to my parents. Your steadfast support, love, and belief in me have been the foundation of my achievements. Thank you for always being there through every challenge and triumph.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CI	Continuous Integration
CI/CD	Continuous Integration/Continuous Deployment
CIS	Center for Internet Security
CNCF	Cloud Native Computing Foundation
CPU	Central Processing Unit
DK	Design Knowledge
DSR	Design Science Research
EDOC	Enterprise Distributed Object Computing
HTML	HyperText Markup Language
ICMP	Internet Control Message Protocol
IT	Information Technology
K8s	Kubernetes
LXC	Linux Containers
LXD	LXC Daemon
MAAS	Metal as a Service
NIST	National Institute of Standards and Technology
OS	Operating System
PDF	Portable Document Format
QA	Quality Assurance
SLA	Service Level Agreement
Sphinx	Python Documentation Generator
TLS	Transport Layer Security
UI	User Interface
VM	Virtual Machine

LIST OF FIGURES

1. INTRODUCTION

This section introduces the purpose and scope of the study. It also provides context and highlights the significance of this capstone research project.

1.1. Project Context

Leveraging cloud capabilities is a key component to optimize modern software solutions, emphasizing continuous integration and continuous deployment (CI/CD), removing silos between development and operations are key to ensure that a project meets competition requirements [1]. Automated functional, security and performance testing, play a key role in ensuring that cloud-native solutions are scalable and meet the quality requirements [2]. Also, documentation is a major component in the development, deployment, and maintenance of cloud-native projects, ensuring clarity and withstanding team changes and refactoring [3]. Additionally, adhering to security hardening practices such as CIS benchmarks is very important to strengthen applications against threats and cyber-attacks [4]. For cloud-native applications, one solution to mitigate these threats could be to integrate security testing in a ci/cd pipeline, which potentially would ensure the discovery of vulnerabilities throughout the development lifecycle [5].

This study investigates the challenges and provides insights on deploying NextCloud on bare metal cloud infrastructure, automated documentation, and tests, including security and functional aspects and integrating them into the CI/CD pipeline. It examines how these measures affect the quality and security of the application, and how CIS hardening improves the overall security of a cloud application. The goal is to ensure a successful deployment and launch on the market. The objective of this study is to ensure that the application meets all standards and best practices regarding security, performance, and functionality. The project is following the DevOps culture emphasizing collaboration and communication between development and operations. This ensures that the whole IT (Information Technology) team is involved in the development, testing and deployment processes through the GitLab tool. It's important to mention that we are not developing the NextCloud application, which is an open-source project, but the development of the test suite. The project is conducted under the CI/CD methodology using GitLab.

1.2. Company Background

The project was conducted at LiJo Technologies, a company specializing in cloud and edge solutions. Founded in 2021, LiJo Technologies is a relatively new actor in the market. It operates as a small start-up with fewer than 10 employees. Despite its size, the company is committed to delivering cloud-based and edge computing solutions. Currently LiJo Technologies is focused on two main solutions: An open-source file storage system based in the Cloud, which is the center of this study and an Edge computing solution.

Located in L'Aouina, Tunis, LiJo Technologies adopts a DevOps culture and emphasizes strong collaboration and communication between team members. The technological stack is built on a bare metal cloud infrastructure managed by Proxmox and Ubuntu virtual machines. The company heavily relies on open-source software primarily from Canonical.

Remote work is facilitated through SSH tunnels, allowing employees to connect to the local office network from home. In this project, the company needed to dedicate a tester to ensure the functionality and security of the NextCloud deployment on a Kubernetes cluster.

1.3. Problem statement

Deploying an application on bare meta cloud environment can be a complex task requiring many components to work together. In an agile process applying continuous integration and continuous deployment, the system is deployed in increments. Running tests with each increment ensures that all components are integrated correctly and work as expected. For the application to be successfully released to the public, we must ensure it meets the standards and requirements of security, functionality, and performance. Data integrity and security should be tested rigorously. Securing such distributed apps that host users' personal data is of the utmost importance. So, practices such as CIS hardening and automating security and conformance checks should be implemented.

1.4. Purpose of the Study

The primary aim of the study is to automate the process of testing the functionalities of the cloud file storage system “NextCloud”. The goal is to deploy the application and integrate the tests into a Continuous integration (CI) pipeline to validate each new increment of the project. This involves creating a well-documented, reusable Python framework containing all the necessary functionalities for testing. Also, to harden the infrastructure and apply conformance tests to assess the system against standards set by the Center of Internet Security and the Cloud Native Computing Foundation (CNCF), and to remediate any deviations. To summarize, the purpose of this study is to test and secure the NextCloud server application in a DevOps automated environment.

1.5. Research Objectives

To breakdown the study’s purpose into several tasks, the project can be divided into the below objectives:

- Deploy the NextCloud application on Kubernetes, to evaluate the scalability and the resilience of the application and finally to identify best practices and challenges.

- Implement an integrated automated test suite in the CI (Continuous Integration) pipeline. These tests ensure that the application is functional and secure.

- Investigate and document how to adapt a python library to cloud-native applications, focusing on cloud architecture, security considerations and performance optimization.

- To provide insights on how these tests can be automated and integrated with a cloud-native solution.

1.6. Research Questions

Research Question 1: How does the integration of automated functional and security tests in a CI/CD pipeline influence the quality and security of NextCloud?

Research Sub-Questions:

1. What measurable improvements in quality and security metrics are observed following the integration of automated testing into NextCloud's CI/CD pipeline?

2. What challenges are associated with integrating these automated tests into the CI/CD pipeline, and how can they be mitigated to optimize NextCloud's deployment?

3. What are the primary benefits realized from integrating automated security and performance testing into the CI pipeline for NextCloud, and how do they balance against the challenges?

Research Question 2: How can CIS hardening be effectively integrated into the development and deployment process of containerized applications to enhance security without negatively impacting key metrics such as deployment time and resource usage?

The rationale behind the research questions

The research questions are defined to explore the impact of integrating automated testing and security measures within the CI/CD pipeline of a cloud-native application. These questions aim to uncover the benefits and challenges associated with such integration. By measuring the improvements in security and performance the study tries to provide measurable evidence on the efficacy of such tests.

1.7. Approach and Boundaries of the Study

1.7.1. Research approach

The study employs a Design Science Research methodology, which involves the iterative process of designing, building, and evaluating artifacts to solve a problem. This approach includes empirical elements where we conduct observations to assess the efficacy of the solution and the effect it introduced to the system. The study specifically focuses on deploying NextCloud application on Kubernetes, integrating automated testing into the CI/CD pipeline, and applying CIS benchmarks for security hardening.

1.7.2. Scope of the study

The scope of this research is limited to the deployment and testing of NextCloud application on microk8s private cloud infrastructure. It focuses on the effect of implementing an automated testing suite for security, and functionality analysis. It also focuses on the application of CIS benchmarks and techniques and their effects on the system's deployment.

1.7.3. Limitations of the study

The study's findings are limited to the local configuration and setup of the cloud environment, where we are using microk8s, ubuntu server VMs. So, they could be generalized to similar configurations but do not ensure the same results.

1.7.4. Delimitations of the study

The study is restricted to a specific cloud infrastructure, Microk8s and Ubuntu Virtual machines, focusing on the integration and impact of automated testing and security hardening measures within this environment. Additionally, it does not cover other cloud platforms or different CI/CD tools beyond the chosen setup.

1.8. Contributions

This study contributes both technically and scientifically by addressing key aspects of automating testing and enhancing security in cloud-native applications. Below are the main technical and scientific contributions of this work.

- **Technical Contributions**: This project contributed technically by creating and integrating a testing suite framework. This framework includes functional and security tests specifically tailored for a cloud application deployed on Microk8s.

  Additionally, it developed and analyzed the process of integrating CIS benchmarks into a CI/CD pipeline. These tests included Ubuntu virtual machines and the Kubernetes cluster and aimed to enhance the overall security of the NextCloud deployment.

- **Scientific Contribution**: The study provided data-driven insights on the impact of CIS hardening on the security and performance of the NextCloud application. It assessed the effect of applying CIS and CNCF best practices and standards on the security of a file storage system.

  It also quantified the challenges and benefits of integrating automated functional and security tests in a real-world cloud deployment. Finally, the project provided a practical example and reference for future deployments and usages in the industry. It highlighted the importance of continuous testing and security compliance.

The project serves as a complete case study on the integration of automated testing and security hardening in a cloud-native environment.

1.9. Structure

In the next sections the study will begin with a background and a literature review on NextCloud, Cloud Computing, DevOps, and Security Testing. Next, the methodology section details the Design Science Research approach including the regulative cycle and validity threats. The work is divided into three iterations covering the initial setup, automated functional testing, and infrastructure security. The last two sections are Results and Findings and Conclusions, where we will dig into answering the research questions, showing the improvements, and discussing the encountered challenges, implications for academia and industry, highlighting enhanced security. In the conclusion, we wil summarize key findings and offer recommendations for future research.

2. BACKGROUND AND LITERATURE REVIEW

In this section we will provide contextual information to be able to understand the main body of the study. It starts with the background and related studies and connects this work to current state of the art and industry practices.

2.1. Background

In this section, we will dive into the concepts and theory that define the capstone project, we will be focusing on the deployment of NextCloud on a private Cloud infrastructure. This study covers the topics of Cloud Computing, DevOps practices, continuous integration, and deployment, and file storage systems. To facilitate understanding this study and its objectives, we would introduce, explain, and define some key concepts such as NextCloud, CI/CD pipelines. This background explains technical terminology and provides a framework to understand and analyze the execution of the project. This section starts with an introduction to NextCloud, the center of this project. Then, it defines the various technologies and tools used, ordered by their usage, from the creation of the infrastructure to the deployment of the cluster and the application to the testing phase.

2.1.1. NextCloud

NextCloud is a suite of client-server software for creating and using file hosting services. It can be hosted in the cloud or on-premises [4]. The project is open source and is maintained by NextCloud GmbH. It is mostly developed with PHP, and one can clone the source code from the official GitHub repository [7]. In the context of this project, the application is deployed on a private, BareMetal cloud, which brings us to the next point of the background section: Cloud Computing.

2.1.2. Cloud Computing

To provide quicker innovation, flexible resources, and economies of scale, cloud computing is the transmission of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the internet, or "the cloud." [5]. In this project's environment, the application is installed on a private cloud, a deployment strategy in which every cloud resource is set aside for a particular client or user group [6]. Since the environment is not yet in a production state, the choice for this model of cloud provides enhanced isolation, security and control over the integrations and deployments. Various tools made it possible to create and administer cloud infrastructure during this project. Here is a list and explanation of the most widely utilized technologies.

- **Proxmox:** A platform for containers and virtual machines to run on. It is entirely open source and built on the Debian Linux platform. [7]. Proxmox is a hypervisor that facilitates the creation, allocation, and commissioning of machines while providing an abstraction for controlling the physical infrastructure. MAAS is used to configure the assigned computers and install the required operating systems and applications, this will be covered in more detail in the next section.

- **Metal as a server (MAAS)**: It is a cloud platform for virtual machines and bare metal server management. It establishes a single point of control for OS images, computers, and networks that are reliable, scalable, and automated. Throughout this project the operating system is Ubuntu server. Once the machines are ready to use, Juju, an orchestration tool enters in play to manage the deployment of the Kubernetes cluster.

- **Juju**: Using charms, Juju is an open-source orchestration engine for software operators that makes it possible to deploy, integrate, and manage applications at any size and on any kind of infrastructure [8]. Within the scope of this project, Kubernetes cluster deployments are orchestrated by Juju. Charms are reusable software packages that contain business logic to automate all aspects of an application's existence. They are used to abstract and facilitate integrations [8].

- **Kubernetes & Microk8s**: Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Services, tools, and support for Kubernetes are widely accessible [9]. Microk8s, a low-ops, minimum production version of Kubernetes, is an extension. It makes containerized applications manageable, scalable, and automated [10]. Throughout this project, it orchestrates the deployment of NextCloud, ensuring it is up and running. This is a very important component of the project since we are deploying the containerized version of NextCloud.

- **Ubuntu and virtualization**: Ubuntu Server, a variant of the Ubuntu operating system created and constructed as the internet's backbone, is the operating system utilized for this project [11]. It is composed mostly of open-source software and is maintained by Canonical. In this context, Ubuntu also supports containerization technologies like LXC and LXD. Running numerous separate Linux systems (containers) on a single control host is possible with LXC (Linux Containers), an operating system-level virtualization technique. With features like snapshots, live migrations, and storage management, LXD is a container hypervisor that offers a more robust and user-friendly interface than LXC. Its goal is to provide a lightweight virtual machine experience by making container management easier. These containers will be used later in the project to provide isolated environments and execute scripts.

The process of provisioning infrastructure, deploying the cluster, deploying the application, and testing it, is automated using GitLab's CI/CD pipelines. Such an approach allows the IT team to focus more on how to improve what they are doing instead of how to do it. Such an approach is what's called DevOps since the operations and the development teams are working together, removing the silos, and emphasizing continuous integration and deployment. This section of the background will focus on the different technologies and terminologies used throughout this project in the DevOps field.

2.1.3. DevOps Approach

DevOps is an approach to remove silos between Operations and Development teams, to improve collaboration and enhance the delivery process [12]. Patrick Dubois first proposed it to settle disputes between developers and system administrators. It increases software quality, facilitates smoother delivery, shortens the time it takes from conception to delivery, and improves communication [12]. It emphasizes continuous integration and continuous delivery, faster feedback, and security [13]. To automate DevOps, GitLab's CI/CD pipelines were used in this project. This section of the background will dive deep into the most important elements used to allow automation of DevOps practices in this research.

- **DevOps**: DevOps describes approaches to speeding up the processes by which an idea goes from development to deployment in a production environment where it can provide value to the user. These approaches require that development teams and operations teams communicate frequently and approach their work with empathy for their teammates. Patrick Debois, considered the father of DevOps, defines it to remove silos between the development and operation to integrate and deploy more frequently. It combines automation, collaboration, fast feedback, and iterative improvement in a way that lets teams make software better, faster, and more cheaply [14]. Many tools are available to leverage DevOps best practices, which brings us to the next point.

- **GitLab**: GitLab is a DevOps tool that collects all the tasks mentioned above under one umbrella, allowing a software development team to accomplish everything with a single tool, using a single GUI, with all the test results and deployment status displayed in a single place [14]. Next, we will discuss the different components of this tool and how they were exploited to leverage the required automation. To conclude, this tool provides the required automation capabilities to abstract deploying and testing applications more frequently. This automation is provided through a GitLab component: CI/CD pipelines.

- **CI/CD pipelines**: A GitLab CI/CD pipeline is a series of steps that are performed on the files whenever the user commits edits to the GitLab-hosted copy of a repository. In this project's context, these scripts perform the infrastructure deployment, application, and tests. These tests include functional tests, security tests and infrastructure tests, the next section will define the key concepts needed to understand what remains of the report.

2.1.4. Testing

The project involves the development of functional and security tests. This section will delve into the key definitions, theoretical concepts and tools needed to better understand the technical complexities of this project. It starts by defining functional testing and its purpose.

- **Functional testing:** Examining individual application functionality areas with the goal of confirming that the features and behavior are in line with the requirements. Stated differently, this kind of testing finds application failures resulting from errors in the implementation of the functional requirements rather than from issues with the program's operating environment [15]. As mentioned at the beginning of this section, NextCloud is an open-source project and is maintained by NextCloud GmbH, thus, correcting bugs in the source code is out of the scope of this project and all tests are following the Blackbox strategy. Which brings us to the next definition.

- **Black Box testing**: A technique that does not require knowledge of software implementation items under test since test cases are designed based on an item's specified or expected functionality. In the context of this project, the aim is to uncover failures in the deployment, meaning that if errors arise, we must investigate how to alter the deployment to correct them. In fact, we are assuming that NextCloud has been rigorously tested and works correctly. To conduct our black box functional tests, we used tools such Selenium to implement a python framework. Next is an overview on Selenium.

- **Selenium**: It is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. It provides extensions to emulate user interaction with browsers (Selenium docs), which is the focus in this project. It was used to simulate human interaction with NextCloud web app and test the core functionalities of the application.

- **Sphinx**: Documentation is a crucial deliverable to ensure scalability, readability, and usability of coding projects. In this project documentation generation is automated using the documentation generator tool Sphinx. It generates PDFs and Web Pages respecting best practices from "restructredText".

Next, we discuss the concepts encountered when implementing the security tests. These tests covered the physical infrastructure, the machines constructing our cluster and the cluster itself. Conformance security tests and remediation fixes are implemented to check and ensure that the system follows CIS benchmarks and CNCF best practices.

2.1.5. Security tests

1. **Conformance testing**: Conformance testing captures the technical description of a specification and measures whether an implementation (i.e., software product or system) faithfully implements the specification. The goal of conformance testing is to provide a level of assurance that the requirements imposed by a specification are being met by implementations claiming conformance to that specification. In this research, the system will be assessed against CIS benchmarks and the CNCF guidelines to evaluate its configurations, integrity, and security [16]. In the next section, we explain the security testing methods implemented in this project.

2. **CIS benchmarks**: The Center for Internet Security defined benchmarks and best practices to ensure correct configuration of IT systems, software, networks, and cloud infrastructure. CIS Benchmarks are developed through a unique consensus-based process involving communities of cybersecurity professionals and subject matter experts around the world [17]. These security tests cover operating systems and Kubernetes clusters and will be applied in this project to harden the Ubuntu virtual machines and the cluster.

3. **Sonobuoy tests**: Sonobuoy is a diagnostic tool that facilitates understanding the state of a Kubernetes cluster by running a set of plugins (including Kubernetes conformance tests) in an accessible and non-destructive manner. It ensures conformance with the Cloud Native Computing Foundation (CNCF) [18]. Sonobuoy also allows testing the following use cases:
    - Workload debugging.
    - Custom data collection via extensible plugins.

2.2. Existing Related Studies

The integration of security, conformance and functional tests within the CI/CD pipelines is important to ensure that deployments are scalable and robust. In this section we will explore existing related studies that focus on the enhancement of cloud-native applications through similar strategies of testing. These papers do not focus directly on NextCloud but have a similar environment, infrastructure, or context. The analysis discusses advancements in automated testing within CI/CD frameworks and the challenges and solutions identified in these studies. This review will serve as a foundation to understand current practices and extract insights.

2.2.1. Integrating Testing in DevOps

The paper "Integrating Testers into DevOps" discusses the important role of testers in the DevOps environment. It emphasizes how to integrate these tests into the continuous delivery model. The study explores both manual and automated testing. It highlights how integrating testers into DevOps improves delivery quality. One of the challenges highlighted by the paper is the cultural shift, where the testers must be integrated in the development process. As later chapters will explain, a similar challenge was encountered in this project. The testers should understand the deployment process fully to test and correct any flaws in the system.
The paper mentions another challenge, tool proficiency and integrating tests without

disrupting project integrations. The following chapters will offer a more comprehensive explanation on how this challenge was treated when deploying NextCloud and its infrastructure. These included an increased deployment time, instances where the tests could not be automated, and situations where the tests could not be integrated into the CI pipeline.

Jayasri S. V. Angara, Srinivas Prasad, and G. Sridevi, published in 2017 a paper "**The Factors Driving Testing in DevOps Setting- A Systematic Literature Survey**" where they reviewed literature to understand the motivational factors that drive testing a DevOps environment. The paper highlighted how DevOps practices impact quality Assurance (QA) and testing functions. The study analyzed literature and academic and industry surveys regarding DevOps testing. It noted an increase of the number publications on DevOps since 2011 and remarkable surge since 2014. Additionally, they stated that there is a significant focus on automation and model-drive development frameworks, a high correlation of DevOps testing with cloud technologies and virtualization. The researchers concluded that this is due to the agility introduced by DevOps, the metric-driven processes, the reduction of complexity and the cost management. The increase in the usage of this new model of testing can be explained by the rapidity and increased frequency of integrations and releases [19]. Like in this project of Testing NextCloud, integrating tests in the deployment pipelines automates the procedure and ensures that every integration is rigorously tested, and failures are automatically remediated. This automation reduces the effort of the teams and redirects their focus on optimizing and improving the system.

To conclude, both papers recognize the important role of automation in DevOps, however, they analyze testing in DevOps from different perspectives. The first paper discusses and emphasizes the integration of testers in the development phase. In contrast, the second paper provides a broader view of the process emphasizing the shift towards automation and model-driven frameworks. The paper "Integrating Testers into DevOps" focuses on how to integrate developers in the development process, how to manage the culture change and how to accelerate the development process, while the 2nd paper is more specific in analyzing the most efficient trends within DevOps testing. Combined these papers offer a complete overview on how to test early and more often and make testers participate more in development process to decrease the time required to deliver functioning code.
Both studies affirm the ongoing evolution of testing in DevOps, they reflect a trend toward more continuous, integrated, and automated testing.

2.2.2. Testing Cloud Platforms

The paper "Framework for testing cloud platforms and infrastructures" addresses the need for reliable testing methods for Cloud platforms and infrastructures. It states that the scalability and distribution of Cloud Computing requires robust testing. The authors of the paper developed a testing framework that creates and executes the test cases. The developed solution introduced improvements in the testing process and enabled faster tests. They noted that these improvements resulted in reduced time for detection and resolution of potential issues. The paper discussed also the encountered challenges when implementing the testing framework, the most important of which are the complexity of Cloud APIs due to its interactive and scalable nature, the lack of standardized testing protocols, the integration with existing systems in the sense that they should not disrupt the ongoing processes and scalability and resource management [20].
When testing the infrastructure on which NextCloud would be deployed (Kubernetes cluster and virtual machines), adherence to CNCF and CIS guidelines and best practices ensured a standardized approach. This project mirrored the principles outlined in the paper by implementing a structured and scalable testing environment. It focused on automating deployment processes and integrating rigorous testing protocols that align with industry standards, thus minimizing the impact on operational continuity and enhancing the overall efficiency of the development lifecycle [20].

Xiaoying, Muyang Li, and Bin Chen discussed the emerging need for specialized testing tools in Cloud computing environments in their paper “Cloud testing Tools”. They reviewed various approaches and tools for testing applications deployed on Cloud environments and addressed their unique challenges. The methodology consisted of conducting surveys of existing tools including research-based and commercially available products. Tools were categorized based on their functionalities, such as multi-layer testing, SLA-based testing, large-scale simulation, and on-demand test environment provision. The study investigates how these could be built directly in Cloud to leverage the capabilities of virtualization. The study found that these tools provide a wide range of functionalities such as automating the setup of the testing environment which speeds up the cycle of testing. Authors stated that future work could focus on automation capabilities to reduce the manual work and reduce the time needed to conduct the tests and enhance the precision and the reliability. This project will focus on automating the usage of Cloud Testing tools, notably Sonobuoy with ensures CNCF conformance and CIS benchmarks. This initiative aligns with the anticipated direction of reducing manual testing effort and enhancing testing accuracy, as indicated by the authors’ projection of future work. By integrating Sonobuoy and CIS benchmarks within a GitLab CI pipeline, the project aims to fully automate the deployment and testing process in a cloud environment, thereby facilitating continuous integration and delivery with a robust emphasis on security and compliance standards.

To review, the two papers investigate testing cloud environments but from a slightly different perspective and methodology. Both papers agree that testing cloud environments is very critical due to its distributed and very complex architecture, yet one discusses a new framework to evaluate the robustness of a cloud application and the second paper is a review of existing tools and how each one tackles different problems and explains, in which context, to use each tool. Additionally, both papers agree that integration is a common challenge, and that manual work is necessary to be able to test a customized environment. Analyzing these two papers, we can reveal the need for a unified framework to test Cloud application, that combines the insights from the analysis of the existing tools, resolving the gaps and challenges, and leveraging the new solutions implemented in the 1st paper.

2.2.3. Integrating Dynamic Security Testing Tools in CI/CD Pipelines

The case study "**Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines**" illustrates the implementation of three automated dynamic testing techniques into a CI/CD pipeline. The paper investigates the trend of DevSecOps, integrating security into DevOps practices so that security does not lag [21].
The study documents each step of the integration process, from the initial setup and configuration to the execution within the CI/CD workflow. The paper evaluates the effectiveness of their methodology by analyzing the security vulnerabilities detected and the overhead added in terms of resource usage and processing time. It states that integrating dynamic security testing tools into CI/CD pipelines can enhance software security and speed up feature delivery but comes with its own challenges. The main challenges encountered are the trade-off between speed and security where the pipeline takes longer time to execute, slowing down the time deploy and release. Additionally integrating the tools to the pipeline and the existing system since they must be configured to identify vulnerabilities and avoid disrupting existing workflows [21].
The exact same difficulties were encountered in this project, where the execution of security takes a considerable amount of time, and cumulated they slow down the integration and deployments pace. Additionally, occasionally they disrupt the pipeline since when tests fail the whole pipeline fails. Overcoming such challenges will be discussed in later chapters of this paper.

The paper "**Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines**" explores the implementation of security tests within an Agile software development lifecycle (SDLC) framework. It focuses on a web-based software system. The testing framework involves a multi stages process using GitLab and Docker. It encompasses continuous development, testing, integration, deployment, and monitoring. The methodology followed in this study centers on automating the build, test, and deployment to shorten the time and execute the operations that were conducted manually previously. They integrated static and dynamic security testing to detect vulnerabilities early in the development process. The results of the paper state a reduction in the deployment times from several hours to 3-4 minutes thanks to the automation of the deployment process. This paper serves as a basis for optimizing the automation of the testing procedure and the optimization of execution time to avoid disruptions. This approach reflects the objectives of the current project, which seeks to refine and enhance automation strategies within CI/CD workflows [22]. In this paper, the challenges addressed in the implementation of the framework are the complexity of the integration, precisely integrating GitLab and Docker and security mechanisms into a cohesive DevSecOps workflow [22]. Also, the security testing overhead introduces a tradeoff between security and performance.

The two studies share similarities, they both have a common goal to reduce deployment times and ensure security. However, there are significant differences in the methodologies and the focus areas, that could invite critique. The first study investigates the impact of using dynamic security testing tools on existing CI/CD workflows and it studies the trade-off between processing time and security. It could have studied static tools such CIS hardening benchmarks which could have provided more security coverage earlier in the development stage. The second study presents a more holistic approach and integrates both dynamic and static security testing. This study only investigates the effects of these automated tests on deployment times but does not focus on its effects on the quality, performance, and resilience of the application. The study could have delved into how this impacts system reliability especially in high-stake environments like cloud services. Deeper analysis could cover more advanced automation techniques that mitigate the overhead introduced by security testing. Also, both studies could strengthen their findings by incorporating and testing real-world cloud native applications that represent a real testing challenge. As an example, a Cloud storage system is a good choice thanks to the critical role of these tests in ensuring data integrity and security.

2.2.4. Literature Insights and NextCloud Testing

The analysis of existing literature on the integration of testing practices within DevOps and focusing on security and performance measures in CI/CD pipelines, represents a strong foundation for this research. The findings highlight the importance of the integration of automated testing to enhance the security and performance of cloud-native applications.

This is particularly relevant in the case of containerized applications such as NextCloud, which especially if deployed in a cloud environment, where CIS hardening and automated testing practices must be integrated without affecting operational efficiency. Tshe literature review identified existing gaps in the existing tools and showed both the potential benefits and challenges associated with these integrations. By mapping these insights against the context of NextCloud, this paper aims to address the critical gaps. For instance, to optimize functional and security testing and to develop a strategy to effectively integrate them in the CI/CD pipeline without disrupting the performance nor affecting the deployment time. In addition, the insights regarding cultural shifts and complexities when integrating will help shape the methodology applied in this paper. To conclude, this literature review contextualizes the body of knowledge and determines the trajectory of the investigation to enhance the robustness and reliability of NextCloud deployments.

4. ITERATIONS

This section details the iterative process followed in the study. In fact, the project's methodology is composed of three interrelated iterations, each building over the conclusions and developments of the previous one. These iterations have been studied to progressively enhance the deployment, security, and functionality of NextCloud application in a private cloud environment using microk8s, a lightweight distribution of Kubernetes. In each iteration we will implement the stages of problem investigation, solution design, design validation, design implementation and solution evaluation.

4. METHODOLOGY

This section explains the strategies and methods used in this study to address the research questions. It also outlines the framework for addressing validity threats to ensure the credibility of the findings.

4.1. Design Science Research

In this section we will introduce and define the methodology used in this paper. We will define Design Science Research methodology and explain how it was applied to answer the research questions. We will present the iterative process we followed to design the artifacts and enhance the deployment and security of the NextCloud application on a private cloud infrastructure.

4.1.1. Definition

Design Science Research (DSR) is a problem-solving paradigm that seeks to enhance human knowledge via the creation of innovative artifacts. Simply stated, DSR seeks to enhance technology and science knowledge bases via the creation of innovative artifacts that solve problems and improve the environment in which they are instantiated. The results of DSR include both the newly designed artifacts and design knowledge (DK) that provides a fuller understanding via design theories of why the artifacts enhance (or, disrupt) the relevant application contexts [23]. In the case, DSR would help us design artifacts aimed at improving NextCloud's CI/CD pipeline in terms of functionality, security, and infrastructure robustness.

4.1.2. The regulative cycle of Design Science Research

The regulative cycle as shown in Figure 1 in DSR guides the development and evaluation of artifacts. It follows a set of stages that we will define in this section:

1. Problem investigation: In this stage we define clearly the challenges associated with automating functional, CIS hardening and infrastructure tests within the NextCloud CI/CD pipeline.

2. Solution Design: In this step we design and develop the testing artifacts which include testing scripts, configuration for CIS benchmarks and infrastructure testing.

3. Design Validation: Validate the designed solutions and ensure that they meet the requirements without disrupting the workflow.

4. Design implementation: Once the testing solutions are validated, we proceed to the implementation in the CI/CD pipeline. We should ensure that they are integrated correctly without disrupting the existing workflow.

5. Solution Evaluation: In this step we will measure the impact of the integrated tests on the deployment cycles. The feedback from this step will enhance future refinement of the testing solutions.

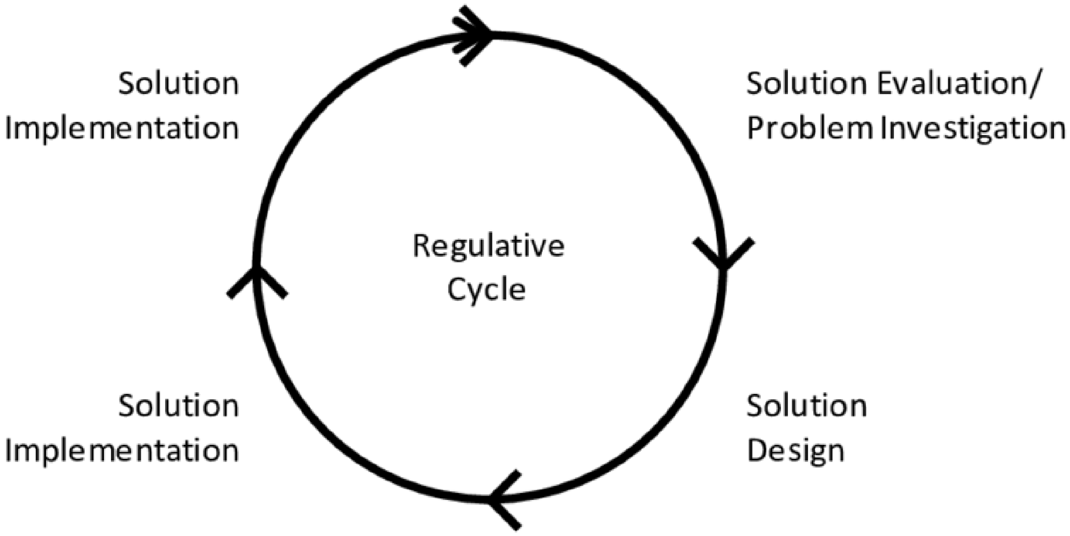


Figure 1: The Regulative Cycle of Design Science Research

4.1.3. Validity threats

As stated in the paper "Introduction to Design Science Research" by Vom Brocke, Hevner, and Maedche, addressing validity threats ensures the credibility of research findings by ensuring that the created artifacts perform as intended in real-world settings. This ensures that the built knowledge is rigorous and relevant [23].

Construct validity threats

Construct validity threats involve ensuring that all constructs that would be used in the study accurately represent what they are supposed to measure. For instance, in the case, in testing NextCloud application, construct validity would involve whether the tests genuinely measure the software's functionality, security and performance. Construct validity is very important because it highlights the relevance of the study to real-world applications.

Internal validity

It focuses on the relationships that we establish in the study. We can consider that internal validity is threatened when the outcomes can be explained by alternative factors. In the case of deploying and testing NextCloud, internal validity would ensure that any observed improvements in the security, performance or functionality of the application are due to the implemented tests and are attributed to the changes made in the deployment process and not to external factors.

External validity

It's often referred to as generalizability. External validity addresses how much the study's findings can be generalized or applied to other contexts. In that case we would try to establish guidelines on how to apply the same methods in testing other cloud native applications like NextCloud. By creating a robust testing framework, we aim to provide a model that can be adapted for similar technologies. This involves detailed documentation of the testing procedures, environment setups, and configurations which could serve as guidelines.

4.2. Iteration 1: Initial setup and environment preparation

4.2.1. Purpose of the iteration

The primary goal of this iteration is to create the infrastructure that would allow the deployment of NextCloud. This includes deploying the hardware resources, setting up operating systems and container orchestration systems, and ensuring that all components are integrated and functioning. Most importantly, this iteration encompasses deploying NextCloud and managing and automating the process through DevOps tools.

4.2.2. Problem investigation

When deploying a Cloud storage solution rigorous testing should be conducted to ensure security, integrity, and functionality. One crucial element that makes the tests reliable is the environment in which these tests are conducted which should closely replicate the production environment. This condition is crucial for implementing accurate tests in later iterations and for detecting any discrepancies early in the process. The main complexities to address to replicate such an environment are the configuration of network settings, storage, and the integration of DevOps tools. Thus, the first problem that this iteration aims to solve is the differences often found between the environments and to deploy NextCloud and its Infrastructure on a bare metal cloud.

4.2.3. Solution design

This section of the solution design starts with explaining the architecture of the system and how they are integrated and interacting together.

- **NextCloud**: NextCloud is a versatile cloud storage system that provides a user-friendly interface for file management accessible via web browsers, as well as Android and iOS apps. As shown in Figure 2 below, users can connect through NextCloud Client which allows additional features such as file synchronization and end-to-end encryption. The NextCloud server is central to the architecture, it manages all user interactions whether through the NextCloud client apps, directly via web browsers, or through WebDAV—a protocol that extends HTTP for collaborative content authoring on web servers [26].

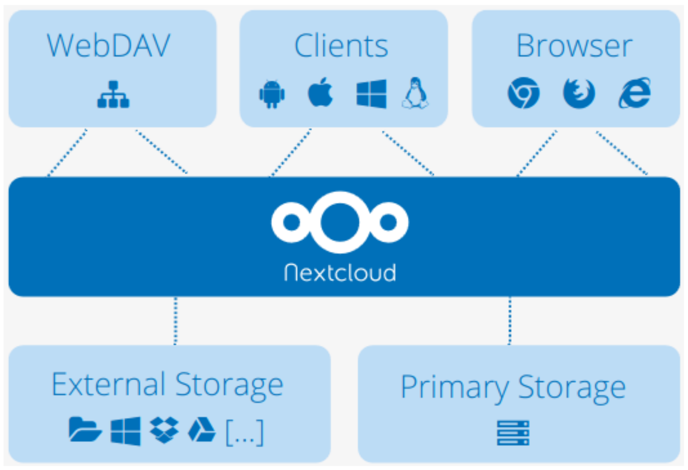


Figure 2: NextCloud Application Architecture [26]

NextCloud Server: The NextCloud Server is the center in the NextCloud architecture, mediating file access and processing while monitoring and logging data access for auditing and analysis using SIEM tools like Splunk. The NextCloud server is configured through a secure web interface, enabling authorized users to control storage, set policies on file access, or set up automated file processing, manage users, enable, or disable functionality and more. The NextCloud Server is a PHP web application, and this project is running on NGINX. It stores file sharing information, user details, application data and configuration and file information in a database [26].

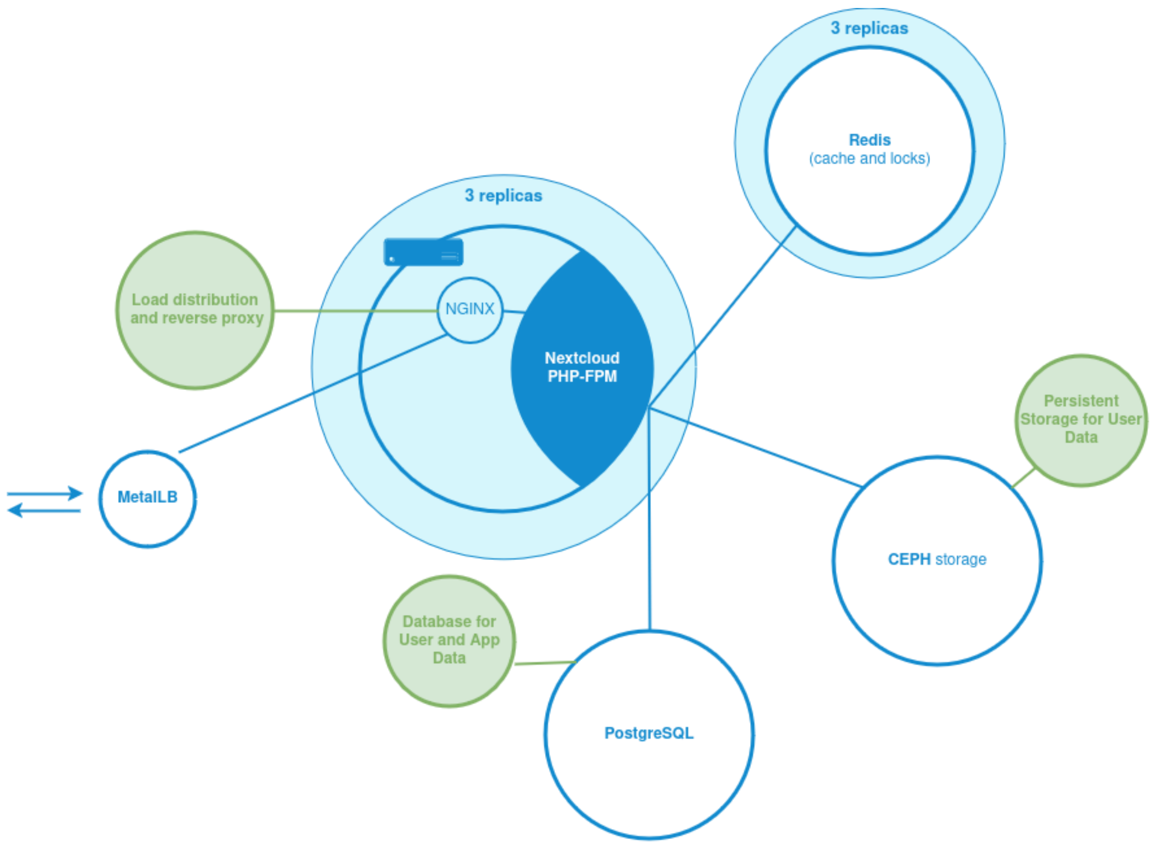

3 replicas
Redis
(cache and locks)
3 replicas
Load distribution and reverse proxy
NGINX
Nextcloud PHP-FPM
Persistent Storage for User Data
MetalLB
CEPH storage
Database for User and App Data
PostgreSQL

Figure 3: NextCloud Deployment Components

NextCloud server uses several services and tools to work as shown in Figure 3 which details its architecture. These tools are:

- **Redis**: Deployed as an in-memory data store, Redis enhances performance by caching frequently accessed data and reduces the load on the database. It is used to handle sessions and file locking. It reduces response times by storing session tokens and file metadata.

- **PostgreSQL**: It is used as the external database for NextCloud. PostgreSQL offers robust data integrity and support for complex queries. The SQL database is responsible for managing all structured data within NextCloud such as user credentials, file metadata and application settings.

- **CEPH**: Employed as the primary storage solution, CEPH offers high performance, reliability, and scalability. In this architecture it is utilized through the Kubernetes persistent volume framework. This integration ensures that data is safe even when hardware fails.

- **NGINX**: Serves as the reverse proxy and load balancer i the architecture, configured to handle HTTP requests and direct them appropriately to the backend NextCloud PHP-FPM service. It enhances the security and efficiency of the server by enabling SSL termination and static content caching.

- **Microk8s cluster**: As mentioned in the background section, microk8s is a lightweight distribution of Kubernetes. In this project the cluster is a collection of ubuntu virtual machines. It serves as the environment in which NextCloud would be deployed. Figure 4 illustrates how the Kubernetes cluster is managed and orchestrated by Juju with multiple nodes supporting NextCloud, Ceph storage and NGINX load balancing. MetalLB enables network load balancing. The setup is explained further in Table 1, it creates a robust scalable and secure environment. Once the environment is ready, the deployment of NextCloud is triggered. Figure 5 offers an architectural high-level overview on the whole system once deployed.

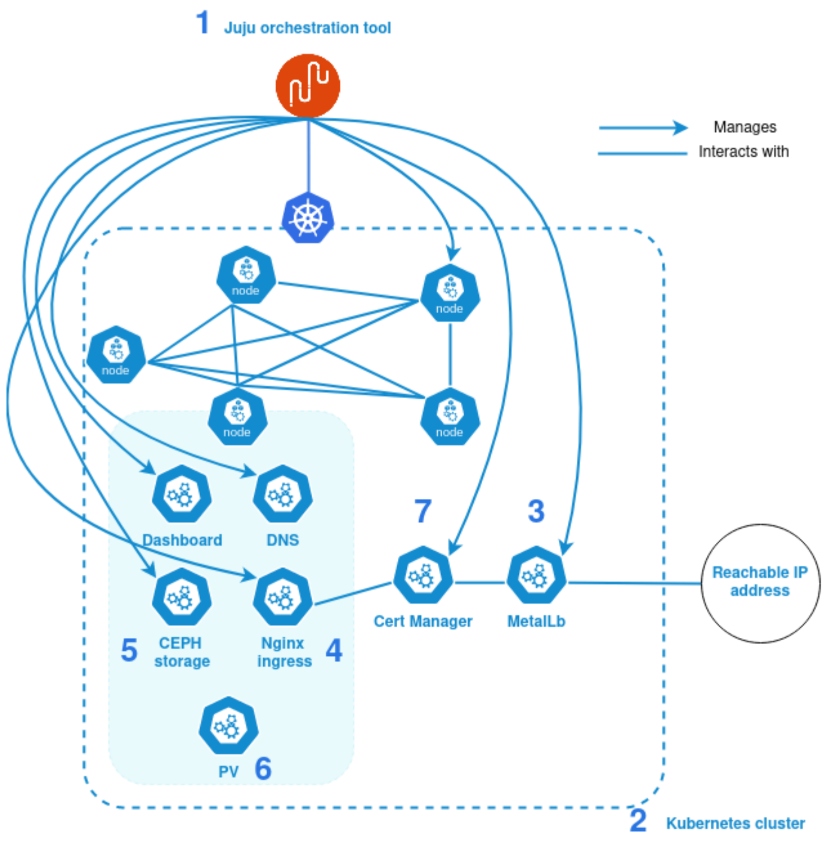


Figure 4: Juju Orchestration of the Kubernetes Cluster

Table 1: Roles and Functions of Key Components in the Kubernetes Cluster

ID	Role	Explanation
1	Juju Orchestration	Juju acts as the orchestration tool that manages the deployment and configuration of Kubernetes clusters and associated services. It interacts with the cloud environment dynamically managing resources according to predefined models and handling failover and scaling.
2	Kubernetes Cluster	Kubernetes serves as the backbone of the architecture, where multiple nodes are configured to support services like Nextcloud, Ceph for storage, and Nginx for load balancing.
3	MetalLB	MetalLB provides load balancing services across the Kubernetes environment, it offers a network load balancer.
4	Nginx: Load balancer and reverse proxy	Nginx Ingress controls access to Kubernetes services from external sources, routing traffic effectively across the cluster's resources.
5	Storage Configuration	Ceph is integrated into the Kubernetes cluster as a storage solution, offering robust, scalable, and secure storage for block, file, and object data. The Ceph components (ceph-mon, ceph-osd, ceph-fs) work together to provide a resilient storage system
6	Persistent volume	Persistent volumes are managed through Kubernetes, with Ceph filesystems utilized to ensure data persistence across deployments and pod restarts.
7	Cert-manager	Cert-manager automates the management of TLS certificates, ensuring encrypted communication and secure access to services.

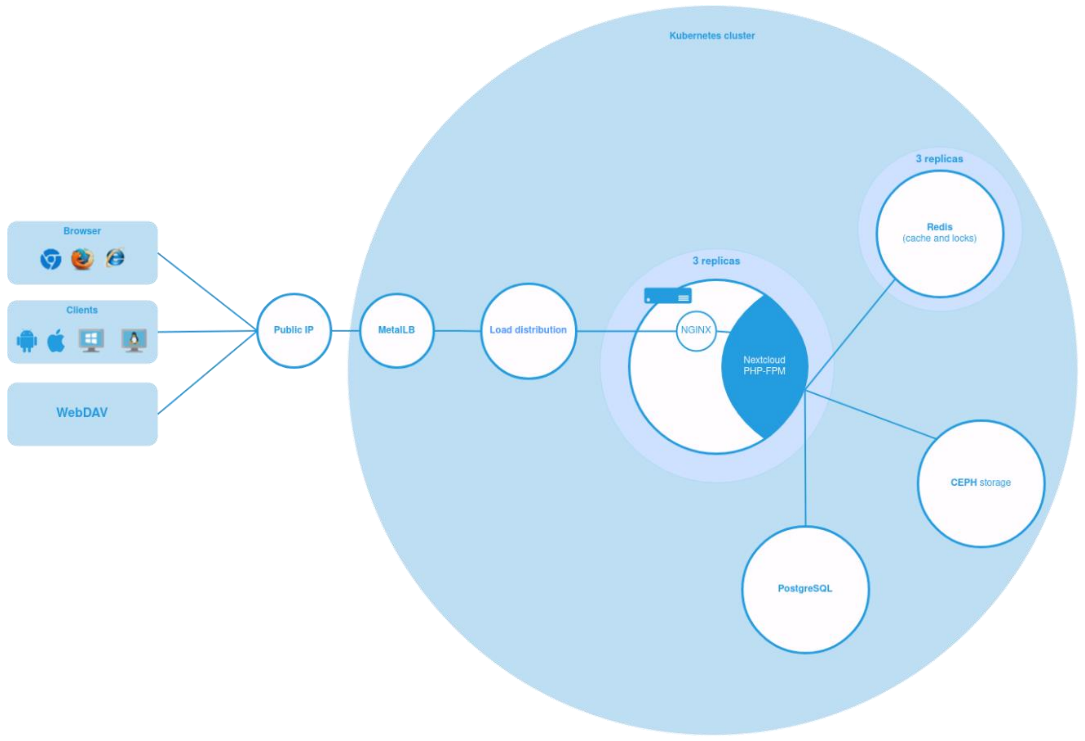


Figure 5: High-Level View of NextCloud on Kubernetes

- **GitLab**: It is the core DevOps platform that enables hosting the scripts and automates the deployment processes through its CI/CD capabilities. Two distinct repositories are hosted, one to deploy the infrastructure and the second to deploy NextCloud using Helm charts. These two directories are:

    - `**charmed-k8s**`: This repository contains all necessary configurations and scripts to set up the Kubernetes infrastructure. It includes all network, storage, and cluster configurations. This is packaged in Helm Charts and will be explained further next.

    - `**nextcloud-k8s**`: Dedicated to the deployment of NextCloud, also using Helm charts, this repository abstracts the deployment process and simplifies the management. Each push to the directories activates the CI/CD pipelines to integrate the updates and execute the deployment with new configurations. These pipelines will be the core of this project since they will be used in all subsequent iterations.

- **Helm charts**: Helm is used in this project to encapsulate and abstract the complexities of deploying both the infrastructure and NextCloud while allowing the developers to alter the configurations and customize them to their environment, as illustrated by Figure 6. It is described as the package manager for Kubernetes. It simplifies the complexities of managing Kubernetes applications by automating the creation, packaging, configuration, and deployment processes. Helm packages, known as charts, are essentially collections of YAML configuration files that describe a related set of Kubernetes resources. These charts are highly reusable, allowing developers to deploy consistent environments, which is essential for DevOps practices [26].

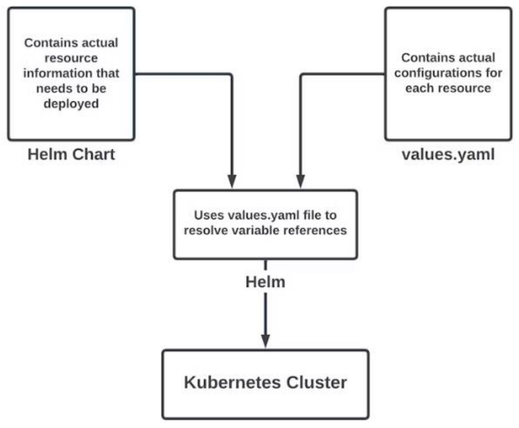

Contains actual resource information that needs to be deployed
Helm Chart
Contains actual configurations for each resource
values.yaml
Uses values.yaml file to resolve variable references
Helm
Kubernetes Cluster

Figure 6: Helm Chart Structure for Kubernetes Deployment

4.2.4. Design validation

When solving a particular problem, design validation refers to evaluating the effectiveness and accuracy of the proposed solution model. As explained in the previous section, it will be assessed against these concepts:

- **Internal validity**: It refers to the degree to which the results obtained can be explained by the studied parameters and no other factors. For the NextCloud deployment, internal validity is achieved by ensuring that variables such as hardware configurations and DevOps tools are consistently controlled.

- **External validity**: The proposed infrastructure and automation processes implemented are generalizable and applicable beyond the specific context of this study. With minor modifications, the solutions and configurations can be adapted for other cloud-native applications.

4.2.5. Solution implementation

The section will explain how both the infrastructure and the application were deployed and will start by detailing the first one. It's important to note that that GitLab and its CI/CD pipelines play a pivotal role in automating all the processes.

Infrastructure deployment implementation

- **Cloud and Kubernetes Orchestration with Juju**

The process begins with the initialization of the Microk8s model with Juju, specifying **`$MICROK8S_MODEL_NAME`** as the model's name. This step uses configurations dedicated to MAAS-managed cloud, in this way Juju orchestrates the deployment.

- **Microk8s and Ceph Cluster deployment**

Microk8s was deployed along with Ceph components (monitors, OSDs, CSI, and file system) to handle both computing and storage needs. The deployment used a predefined bundle `bundle.yaml` to abstract the setup and apply to multiple services.

- **Networking with MetalLB and Ingress**

To distribute external traffic to services within the cluster, MetalLB is configured as the load balancer. Additionally, NGINX is installed as the ingress controller to manage routing and SSL termination.

- **Certificate management with cert-manager**

Cert-manager was set up to automate the provisioning and management of TLS certificates. It is very important for securing communication with NextCloud once deployed. A Cluster issuer was configured to integrate with Let's encrypt to allow the renewal of certificates.

- **Automation with GitLab CI/CD**

The automation of the deployment and management processes for the Microk8s-based infrastructure is allowed using the continuous integration, continuous deployment pipelines of GitLab. This approach ensures consistent and error free deployments. The pipeline as illustrated in Figure 7, is configured through the `**gitlab-ci.yaml**` file defines the `**create**` stage and in later iterations the testing stages. These pipelines not only automate the deployment process but also ensure that any changes to the infrastructure or application code are systematically deployed without manual intervention.

NextCloud deployment implementation

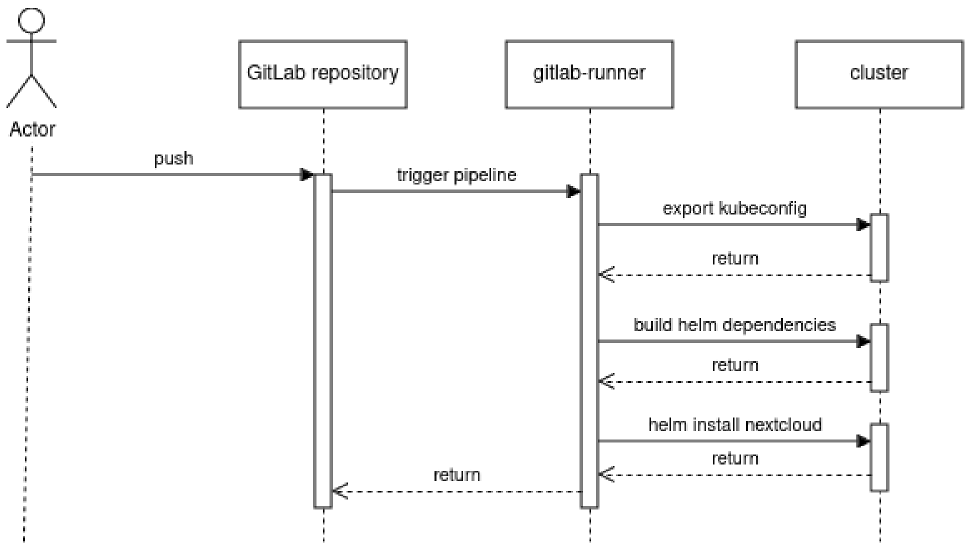


Figure 7: CI/CD Pipeline Sequence Diagram

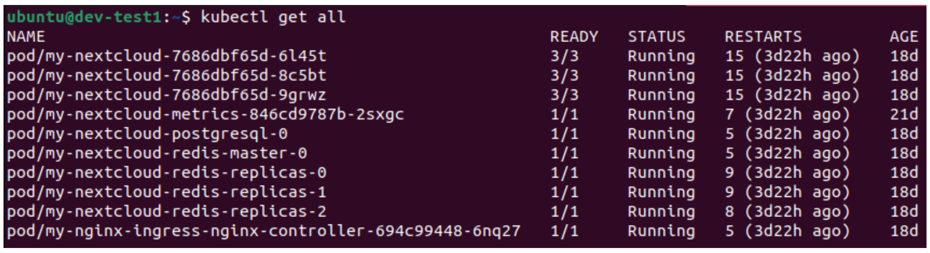
As explained in the sequence diagram, the deployment of NextCloud within the Kubernetes environment is automated through the GitLab CI/CD pipelines. The process starts by setting the `**KUBECONFIG**` environment variable within the GitLab runner. This variable directs command line tools to the correct Kubernetes cluster configuration and ensures that all operations are performed in the intended cluster context.

The deployment is executed using Helm Charts. Helm simplifies Kubernetes application management and installation. It allows abstract customization through the `**values.yaml**`. The file serves as the primary method to alter the behaviors of the Helm chart and, by extension, the Kubernetes resources it manages. Its values are defined to override the defaults set within the Helm chart templates. Helm comes with a default predefined file containing all possible configurable parameters.

The key configurations include:

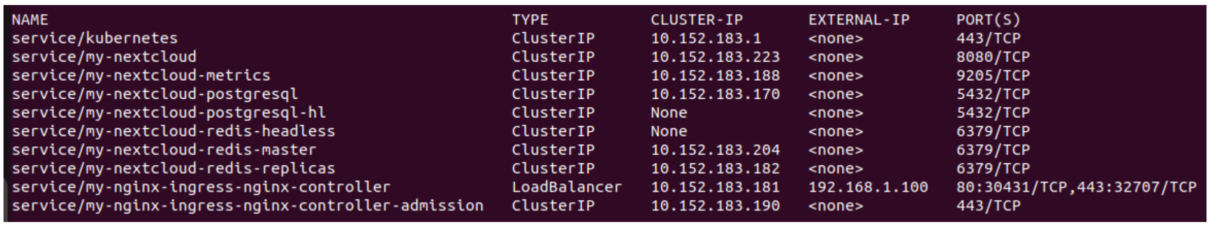
- **Image configuration:** We define the official NextCloud Image and its version.
- **Replica Count:** We define how many replicas of the NextCloud to deploy and define high availability and load distribution.
- **Ingress configuration:** We enable ingress, set its type to Nginx, and define other preferences.
- **Persistence settings:** We enable data storage persistently across pod recreations, we set the storage class to CEPH and define the access mode.
- **Database Configuration:** We enable the external database option and set it to point to PostgreSQL and define credentials.
- **Security Contexts:** We ensure that pods run with appropriate permissions and security settings.

After the execution of the pipeline script, developers can check the installation of the various resources through the Kubernetes command line tools as shown in Figure 8, Figure 9 and Figure 10.



```
ubuntu@dev-test1:~$ kubectl get all
NAME                                              READY   STATUS    RESTARTS         AGE
pod/my-nextcloud-7686dbf65d-6l45t                 3/3     Running   15 (3d22h ago)   18d
pod/my-nextcloud-7686dbf65d-8c5bt                 3/3     Running   15 (3d22h ago)   18d
pod/my-nextcloud-7686dbf65d-9grwz                 3/3     Running   15 (3d22h ago)   18d
pod/my-nextcloud-metrics-846cd9787b-2sxgc         1/1     Running   7 (3d22h ago)    21d
pod/my-nextcloud-postgresql-0                     1/1     Running   5 (3d22h ago)    18d
pod/my-nextcloud-redis-master-0                   1/1     Running   5 (3d22h ago)    18d
pod/my-nextcloud-redis-replicas-0                 1/1     Running   9 (3d22h ago)    18d
pod/my-nextcloud-redis-replicas-1                 1/1     Running   9 (3d22h ago)    18d
pod/my-nextcloud-redis-replicas-2                 1/1     Running   8 (3d22h ago)    18d
pod/my-nginx-ingress-nginx-controller-694c99448-6nq27   1/1     Running   5 (3d22h ago)    18d
```

Figure 8: NextCloud Kubernetes Pods Status



```
NAME                                               TYPE          CLUSTER-IP      EXTERNAL-IP     PORT(S)
service/kubernetes                                 ClusterIP     10.152.183.1    <none>          443/TCP
service/my-nextcloud                               ClusterIP     10.152.183.223  <none>          8080/TCP
service/my-nextcloud-metrics                       ClusterIP     10.152.183.188  <none>          9205/TCP
service/my-nextcloud-postgresql                    ClusterIP     10.152.183.170  <none>          5432/TCP
service/my-nextcloud-postgresql-hl                 ClusterIP     None            <none>          5432/TCP
service/my-nextcloud-redis-headless                ClusterIP     None            <none>          6379/TCP
service/my-nextcloud-redis-master                  ClusterIP     10.152.183.204  <none>          6379/TCP
service/my-nextcloud-redis-replicas                ClusterIP     10.152.183.182  <none>          6379/TCP
service/my-nginx-ingress-nginx-controller          LoadBalancer  10.152.183.181  192.168.1.100   80:30431/TCP,443:32707/TCP
service/my-nginx-ingress-nginx-controller-admission ClusterIP    10.152.183.190  <none>          443/TCP
```

Figure 9: NextCloud Kubernetes Services Status

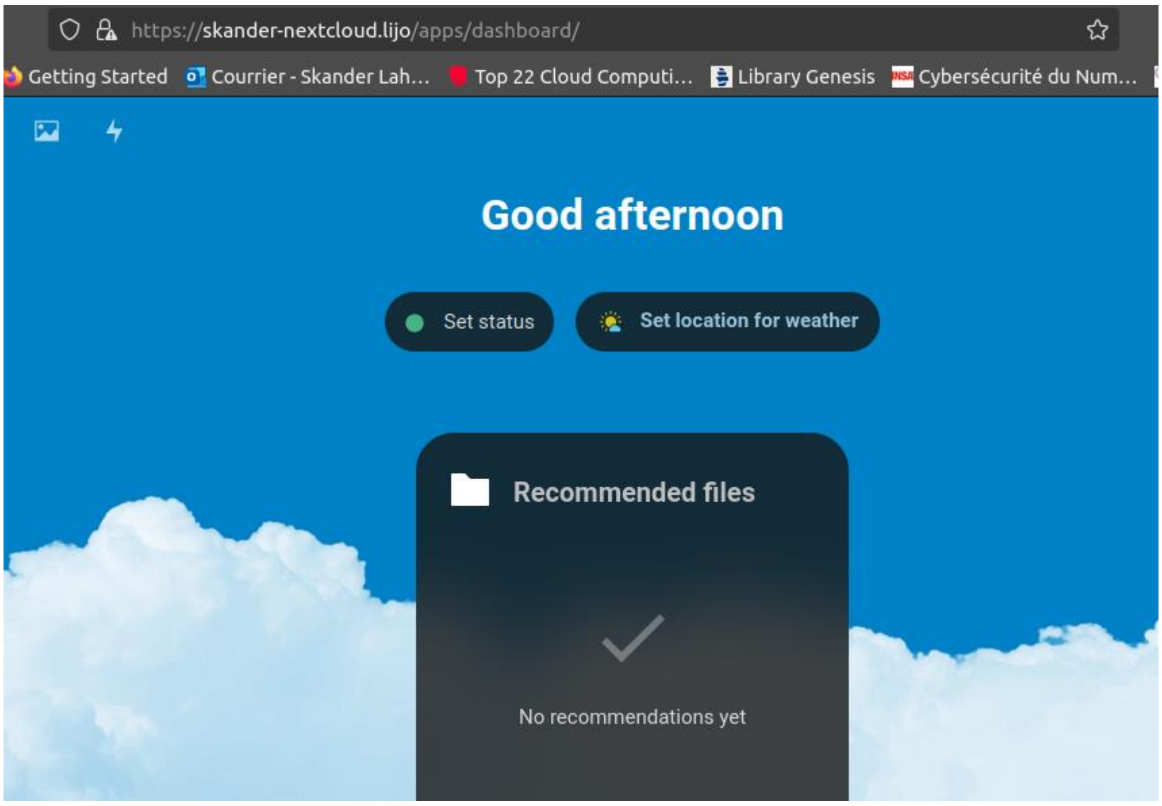


Figure 10: NextCloud Dashboard Interface

4.3. Iteration 2: Integration of automated functional tests

4.3.1. Purpose of the iteration

The focus of this iteration is on developing an automated functional testing framework to ensure the robustness and reliability of NextCloud's core functionalities. This phase involves creating and implementing automated tests to verify core functionalities such as user management, file handling, and end-to-end encryption. These tests are integrated into the CI/CD pipeline to facilitate early detection and resolution of issues.

4.3.2. Problem investigation

Once the application is successfully deployed and we are sure that the scripts to automatically deploy NextCloud through the CI pipeline work correctly, we need to ensure that the core functionalities of the system are functioning. For that, functional tests are integrated into the pipeline to be executed after each new deployment so that errors are detected very early in the process. These tests should be automated to shorten the time required to deploy and to allow developers to correct any revealed bugs before the application is released.

For that, we need to define the testing framework and scenarios. The framework should include a set of test cases that cover the most important functionalities according to the stakeholders. Additionally, the tests should be platform-independent, which means the solution should ensure consistency and reliability across different environments.

4.3.3. Solution Design

This section presents the testing framework designed for NextCloud in this project. The framework aims to ensure the reliability and functionality of NextCloud through automated test cases covering user management, authentication, file management, quota management and end-to-end encryption. In this section, we will present the use cases of NextCloud and the test case scenarios of the testing framework.

Framework overview

The framework aims to simulate user interactions and verify the core functionalities of NextCloud. It uses Python and Selenium for their abilities to simulate browser-based tasks and interact with web applications. Users can upload, download, and delete files. Additionally, they can share files with other users, create other users with different permissions and enable end-to-end encryption to protect their data against the server. The primary objectives of the testing framework will be to:

- To verify the correctness of user management operations.
- To ensure reliable authentication mechanisms.
- To validate the integrity and accessibility of file management operations.
- To confirm the accurate application of user quotas.
- To test the effectiveness of end-to-end encryption.

The testing framework is structured as follows:

Table 2: Directory Structure for NextCloud Functional Testing Framework

Directory/File	Description
functional_testing/	Main directory containing all test-related scripts, utilities and configurations.
├── **cleanup/**	Contains scripts for cleaning up test artifacts post-testing.
│ ├── delete_test_files.py	Script to delete test files created during tests.
│ └── init.py	Initialization file to make Python treat the directory as a package.
├── **config/**	Holds configuration scripts and environment variables.
│ ├── configuration.py	Main configuration file setting up environment variables and other settings.
│ ├── init.py	Initialization file for the config package.
│ └── requirements.txt	Lists all Python dependencies required by the project.
├── created_files/	Directory where files created during testing are stored (not listed in tree output).
├── **docs/**	Documentation files explaining the testing setup and codebase.
├── downloads/	Directory for storing downloaded files during tests.
├──**file_management_cycle.py**	Script importing and executing all tests. Would later be executed in **setup.sh**
├── **generate_documentation/**	Contains scripts to automatically generate documentation from code comments.

│ │ ├── generate_doc.sh	Shell script to run Sphinx and generate documentation.
│ └── modify_conf.py	Script to set documentation configuration settings.
├── media/screenshots/	Store screenshots used in documentation.
├── readme.md	Markdown file providing an overview of the project and how to use it.
├── **setup.sh**	**Shell script to set up the testing environment and run all tests.**
├── **tests/**	Contains all test modules designed to run against the NextCloud instance.
│ ├─end_to_end_encryption.py	Tests end-to-end encryption functionalities of NextCloud.
│ ├── init.py	Initializes the tests directory as a Python package.
│ ├── test_create_user.py	Script to test user creation functionality.
│ ├── test_delete_user.py	Script to test user deletion functionality.
│ └── (other test scripts)	Additional test scripts for various functionalities like file upload/download, integrity checks, etc.
├── **utils/**	Utility scripts that support various functionalities within the tests.
│ ├── file_creation.py	Contains subfunctions involved in file management.
│ └── (other utility scripts)	Other supporting scripts such as user management and WebDriver setup.
pyproject.toml	Contains build tool settings and dependencies for Python projects.

Test case scenarios

- **End-to-End Encryption Tests (`end_to_end_encryption.py`):** End to end encryption ensures that clients data is protected against the server, so that only the owner of the data can access and use it. By “against the server” we mean that even the maintainers of the environment on which NextCloud is deployed cannot have access to user's files.

- **User Management Tests:**
    - **Create User (`test_create_user.py`)**: Tests the user creation process, ensuring that a new user can be added with appropriate credentials and permissions.
    - **Delete User (`test_delete_user.py`)**: Ensures that users can be deleted along with their data and are no longer accessible nor present on the system.

- **File Management Tests**
  - **File Upload (`test_file_upload.py`)**: Checks the ability to upload files to NextCloud and verifies their presence in the list.
  - **File Download (`test_file_download.py`)**: Tests downloading files and verifies that they are correctly saved to the local system.
  - **File Delete (`test_file_delete.py`)**: Ensures files can be deleted from NextCloud and checks they are no longer present in the system.
  - **File Integrity (`test_file_integrity.py`)**: Verifies that files are not damaged when uploaded and downloaded by comparing hash values of uploaded vs downloaded.
  - **File Sharing (`test_file_sharing.py`)**: Tests the functionalities of sharing a file internally with another NextCloud user and externally with anyone having a link that points to the file. Figure 11 illustrates the workflow of this scenario.

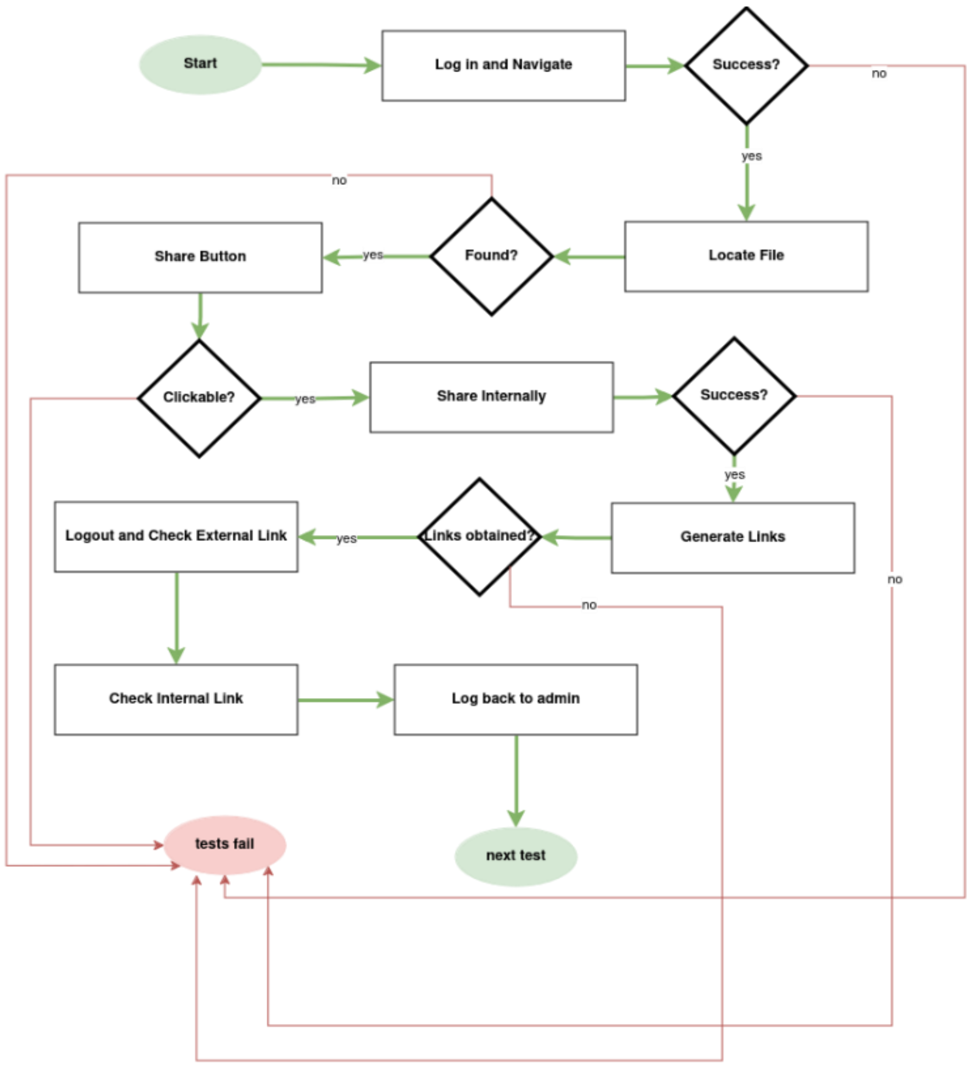


Figure 11: Use Case Scenario – File Sharing Functionality

- **Session Management Tests**
    - **Login (`test_login.py`):** Verifies that a user can login into NextCloud using correct username and password. Figure 12 below explains the workflow of this test case scenario.

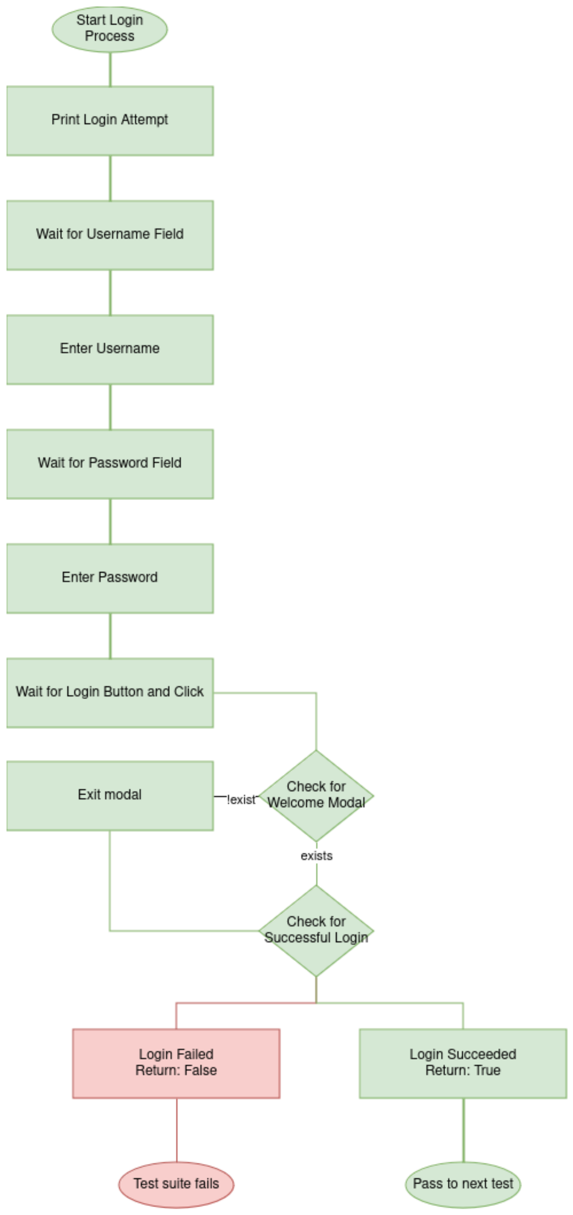


Figure 12: Use Case Scenario – User Login Functionality

    - **Logout (`test_logout.py`):** Checks that the user can logout and checks for the deletion of the session token and the appearance of the login screen upon the execution of the logout action.

- **Quota Management Test (`test_modify_quota.py`)**: This test adjusts the quota allocated to the user and then checks if it was updated by uploading a file that is larger than the previous quota.

Repository organization

A new repository has been created to host the testing scripts. This isolation simplifies updates, enhances security, and improves reusability of the tests. It would be cloned whenever tests need to be executed in the **`gitlab-runner`** deploying NextCloud. The tests would be later executed in an LXC container. We will cover this in more detail in the next sections.

4.3.4. Design validation

In the design validation phase, three critical questions are addressed to evaluate the effectiveness and applicability of the proposed automated functional testing framework:

Internal Validity

The internal validity of the framework is ensured by rigorously defining and aligning test scenarios with the intended objectives. The automated tests are designed to simulate user interactions accurately, ensuring that functionalities such as user management, file handling, and end-to-end encryption operate as expected. Standardized testing protocols and guidelines are followed consistently, ensuring the uniformity and reliability of the test outcomes. This meticulous approach validates that the automated tests can effectively identify and resolve issues within the NextCloud application.

Trade-offs

One significant trade-off encountered in this iteration is the balance between test coverage and the time required for test execution. While automated testing enhances the efficiency of identifying issues early in the deployment process, it requires substantial initial setup and maintenance effort. Additionally, there is a potential trade-off between the thoroughness of automated tests and the nuances of manual testing, which might capture certain edge cases and user-specific scenarios. This trade-off necessitates a careful consideration of test scope and the inclusion of periodic manual testing to complement the automated framework.

External Validity

To ensure the external validity of the testing framework, the design and implementation processes are generalized to apply beyond the specific context of the NextCloud deployment. The framework's architecture is designed to be adaptable, allowing its application to other cloud-native applications requiring similar functional validations. A configuration file allows modifying paths to all web elements, and the functions are all modular and can be modified. This broad applicability ensures that the testing framework can be a valuable tool in diverse deployment environments, reinforcing its relevance and utility in real-world scenarios.

4.3.5. Solution implementation

In section we dive into the implementation of the second artifact of the Design Science Research: The automated testing framework for NextCloud. This artifact represents a suite of Python functional tests created to ensure the integrity, security, and functionality of the NextCloud deployment. We will break down the testing components, detailing the framework's architecture and different test scenarios. Through this artifact we aim to demonstrate a replicable model for automated testing in cloud-native applications.

Functional tests

The solution is to create a framework for Python tests. These tests are designed to test the core functionalities of the application to ensure a reliable cloud storage system. The testing framework developed using Selenium simulates typical user actions to verify that every aspect of the NextCloud server operates as we intended. The test case scenarios are structured into separate modules, each encapsulating specific functionalities to enable granular testing. This modular architecture facilitates the creation of a comprehensive test suite, where each module can be imported individually or combined to execute a series of tests in a controlled sequence.

The centralization of test execution is achieved using **`file_management_cycle.py`** module. This module serves as the orchestrator for the test suite, ensuring that all test scenarios are executed in a predefined order. It is designed to halt the test sequence upon encountering a failure, thereby providing immediate feedback on issues. By initiating this module, we can perform a full suite of functional validations across the NextCloud platform with a single command. This approach simplifies the testing process and enhances its reliability by verifying each component systematically.

To automate the procedure and ensure the environment is ready to execute the **`file_management_cycle.py`** module and therefore the whole test suite, another layer of abstraction is added. A bash script, **`setup.sh`**, installs all necessary packages, executes the python module, then cleans up, as shown in Figure 13. The script is structured to facilitate a full cycle of test setup, execution, and cleanup within a CI pipeline to ensure that the environment is appropriately configured and that dependencies are managed effectively. The execution follows the following steps:

1. **Error Handling and Debugging:** Execute **`set -euxo pipefail`** to ensure that the script exists on failure and that all commands are logged.
2. **Dependency Management:** Updates package list and installs necessary dependencies such as Python, Selenium, Chrome, and Chrome driver.
3. **Python environment Setup:** Creates Python virtual environment and installs required packages.
4. **Environment Configuration:** Configures environment variables such as ChromeDriver path, target URLs and user credentials.
5. **Python Library Build:** Builds and installs the local Python library.
6. **Test Execution:** Executes the **`fil_management_cycle.py`** script.
7. **Cleanup Function:** Ensures that after tests are completed or if an error occurs the environment is cleaned up by removing installed packages, deleting downloaded files, and unsetting environment variables.

Since tests are executed in the CI/CD pipeline, running on Ubuntu Server we have no access to the User Interface, the tests are run in headless mode. But for the purpose of the experimentation, and to show how the tests are executed one test has been run using Ubuntu server to show the simulation of the user interaction as shown in Figure 14.

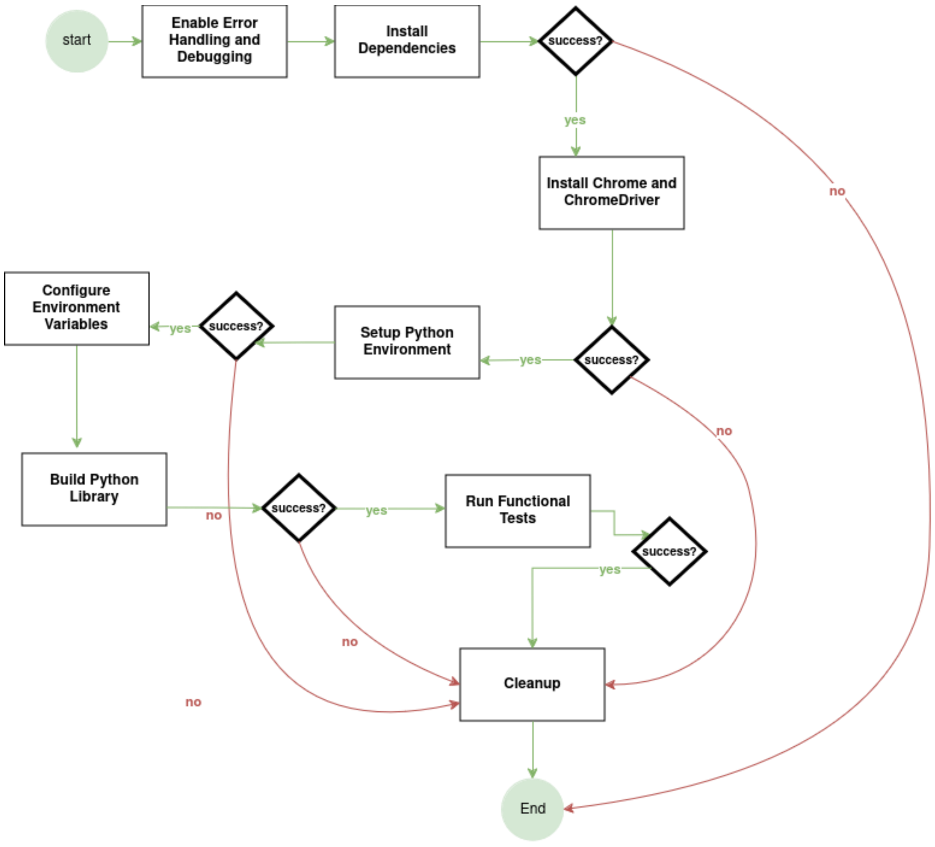
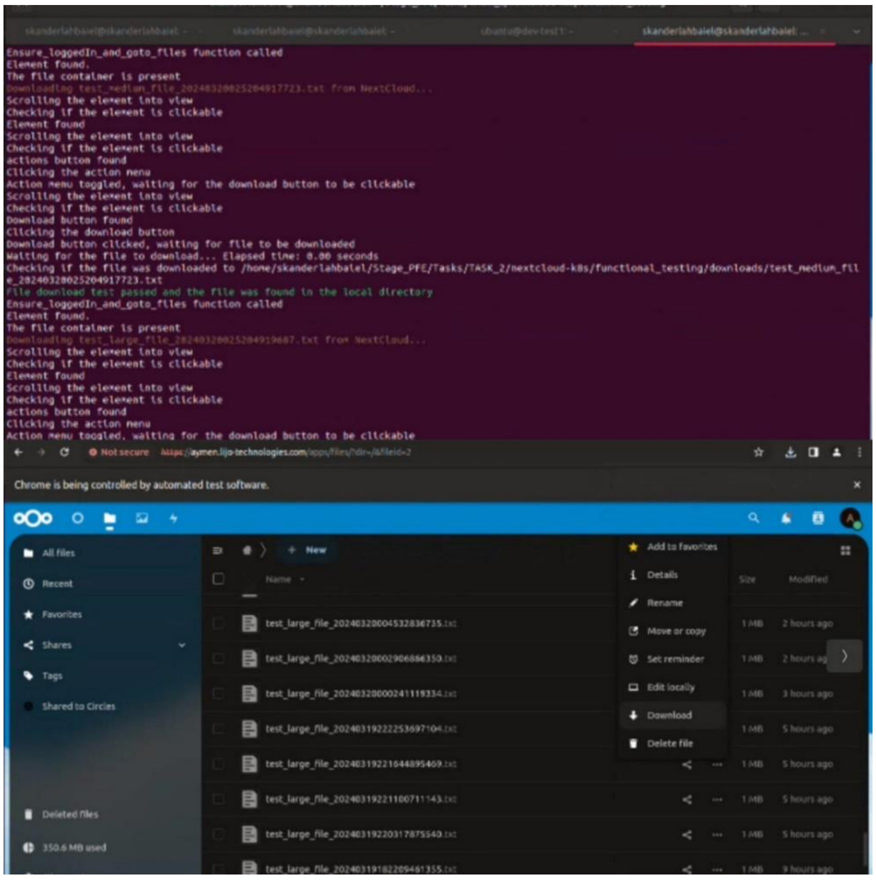


Figure 13: Functional Test Execution Flow



```
Ensure_loggedIn_and_goto_files function called
Element found.
The file container is present
Downloading test_medium_file_20240320025204917723.txt from NextCloud...
Scrolling the element into view
Checking if the element is clickable
Element found
Scrolling the element into view
Checking if the element is clickable
actions button found
Clicking the action menu
Action menu toggled, waiting for the download button to be clickable
Scrolling the element into view
Checking if the element is clickable
Download button found
Clicking the download button
Download button clicked, waiting for file to be downloaded
Waiting for the file to download... Elapsed time: 0.00 seconds
Checking if the file was downloaded to /home/skanderlahbaiel/Stage_PFE/Tasks/TASK_2/nextcloud-k8s/functional_testing/downloads/test_medium_fil
e_20240320025204917723.txt
File download test passed and the file was found in the local directory
Ensure_loggedIn_and_goto_files function called
Element found.
The file container is present
Downloading test_large_file_20240320025204919687.txt from NextCloud...
Scrolling the element into view
Checking if the element is clickable
Element found
Scrolling the element into view
Checking if the element is clickable
actions button found
Clicking the action menu
Action menu toggled, waiting for the download button to be clickable
```

Figure 14: Automated Testing Output

Integration in the CI/CD pipeline

The tests are executed in the continuous integration pipeline of the nextcloud-k8s repository. Thus, we need to understand how the stages are implemented. In addition to the `***deploy***` stage explained in the first iteration, two other stages are created: `***setup***` and `**test**` stages and are to be executed just after the deployment terminates in a success. These two new stages are responsible for setting up an LXC container inside the `**gitlab-runner**` and then cloning the testing repository to the container and executing the tests. The whole process is illustrated in Figure 15.

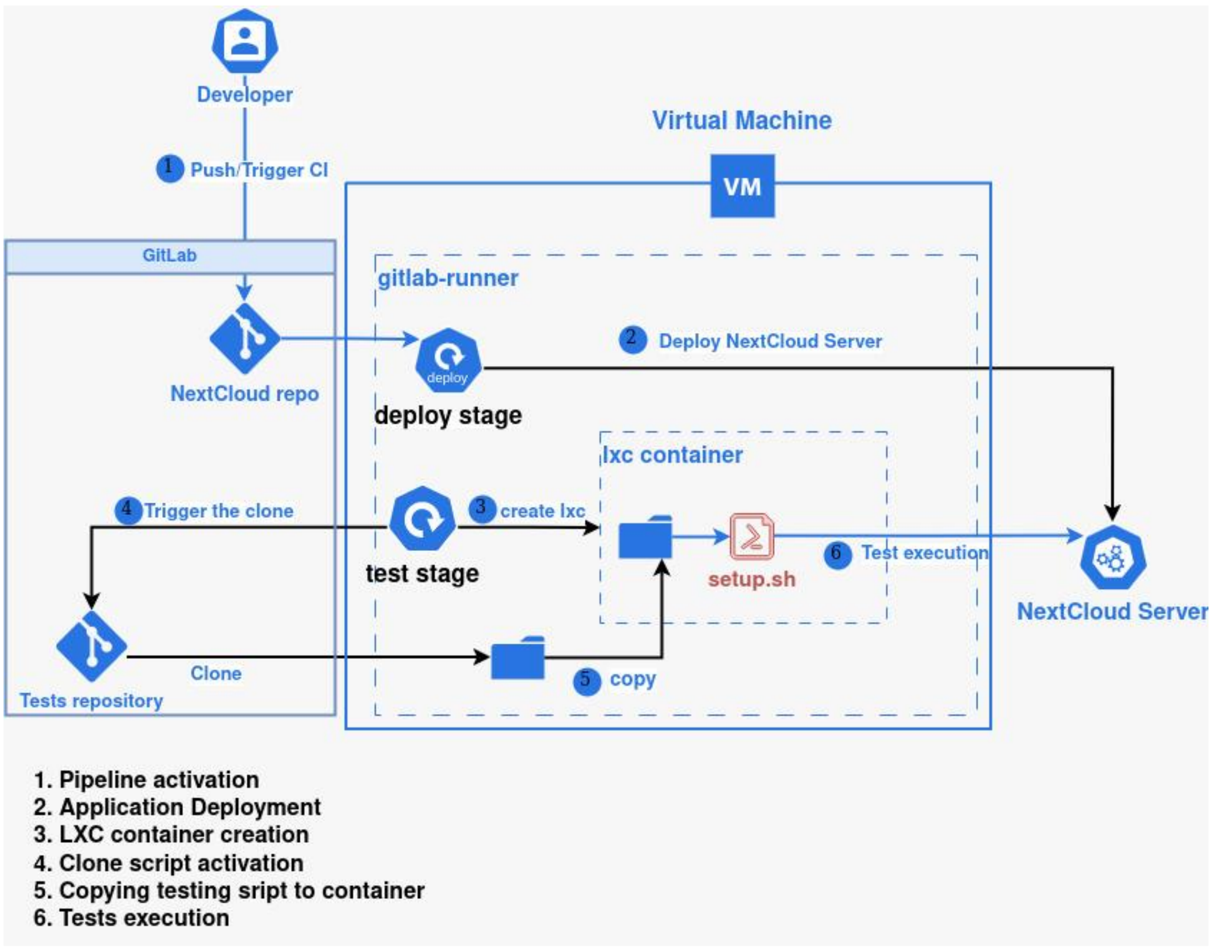


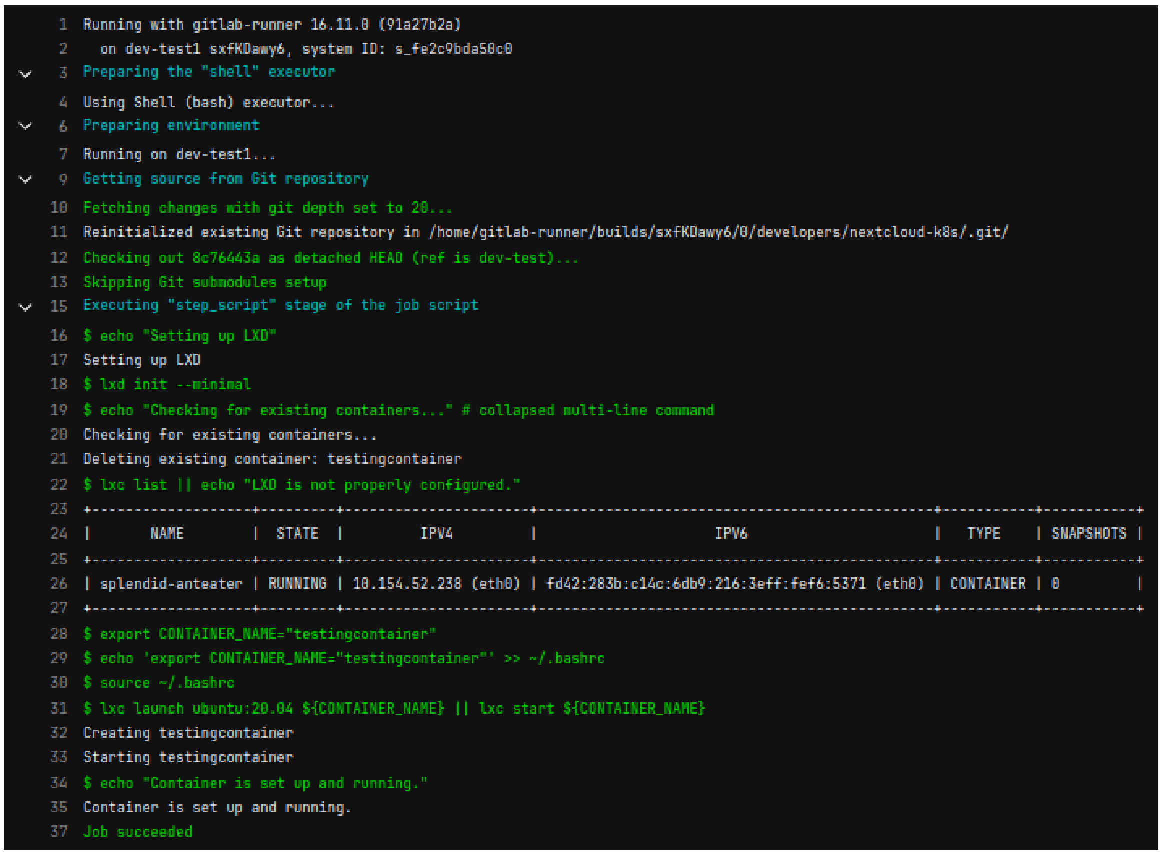
Figure 15: CI/CD Pipeline for NextCloud Testing

The choice behind LXC is due to its nature as a powerful system container and virtual machine manager. It provides a unified experience for running and managing full Linux systems inside containers or virtual machines. Supporting images for many Linux distributions, it scales from one instance on a single machine to a cluster in a full data center, making it suitable for running workloads both for development and in production [27].

These containers are temporary, and they can be created, deleted, and replaced easily as needed [13]. LXC provides a closer approximation to a real server or VM environment, including running multiple processes and services. In this project they enable consistency when replicating the environment in which functional tests will be executed. As depicted in Figure 15 the process follows these steps:

1. Trigger the pipeline.
2. Deploy NextCloud.
3. Create and launch the container and install ubuntu.
4. Clone the tests repository to the `**gitlab-runner**`.
5. Push the tests to the container.
6. Execute the tests by running `***setup.sh***`.

Cloning is done to the GitLab runner and not directly to the LXC container for security and optimization reasons. This approach prevents the exposure of credentials and optimizes the testing process. It also ensures a consistent replicable environment in which we execute tests. Figure 16 shows the output of the GitLab runner after creating the container.



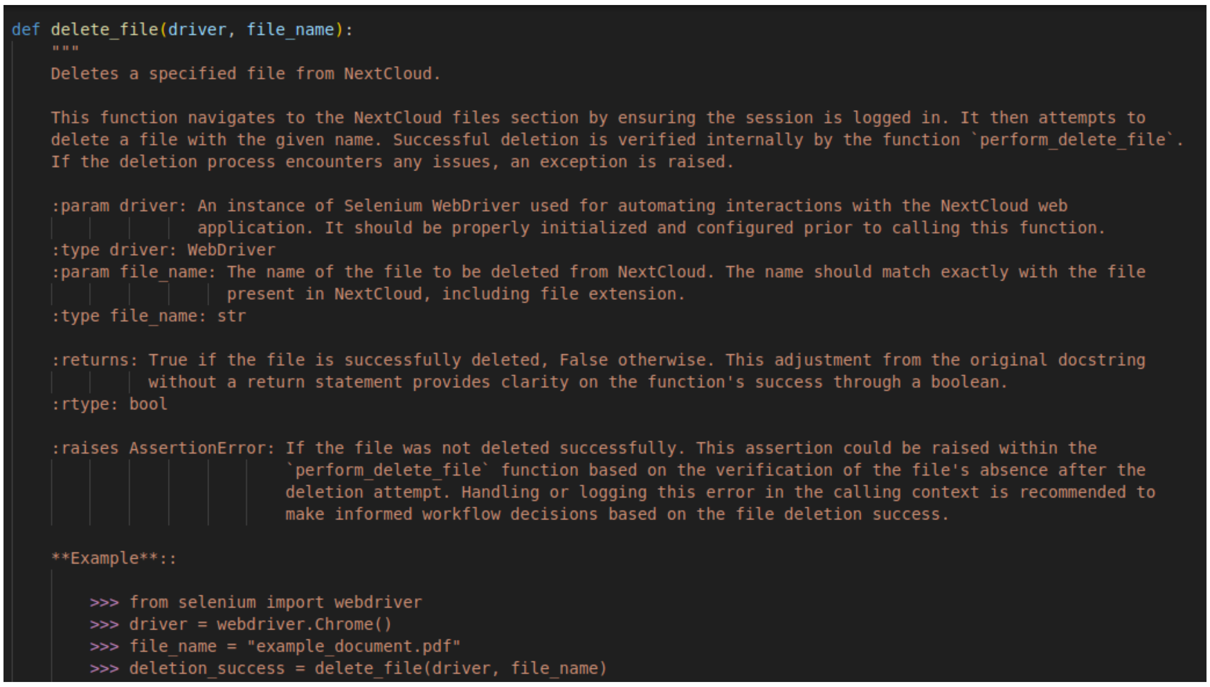
```
Running with gitlab-runner 16.11.0 (91a27b2a)
  on dev-test1 sxfKDawy6, system ID: s_fe2c9bda50c0
Preparing the "shell" executor
Using Shell (bash) executor...
Preparing environment
Running on dev-test1...
Getting source from Git repository
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /home/gitlab-runner/builds/sxfKDawy6/0/developers/nextcloud-k8s/.git/
Checking out 8c76443a as detached HEAD (ref is dev-test)...
Skipping Git submodules setup
Executing "step_script" stage of the job script
$ echo "Setting up LXD"
Setting up LXD
$ lxd init --minimal
$ echo "Checking for existing containers..." # collapsed multi-line command
Checking for existing containers...
Deleting existing container: testingcontainer
$ lxc list || echo "LXD is not properly configured."
+-------------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
|       NAME        |  STATE  |         IPV4         |                     IPV6                      |   TYPE    | SNAPSHOTS |
+-------------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| splendid-anteater | RUNNING | 10.154.52.238 (eth0) | fd42:283b:c14c:6db9:216:3eff:fef6:5371 (eth0) | CONTAINER | 0         |
+-------------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
$ export CONTAINER_NAME="testingcontainer"
$ echo 'export CONTAINER_NAME="testingcontainer"' >> ~/.bashrc
$ source ~/.bashrc
$ lxc launch ubuntu:20.04 ${CONTAINER_NAME} || lxc start ${CONTAINER_NAME}
Creating testingcontainer
Starting testingcontainer
$ echo "Container is set up and running."
Container is set up and running.
Job succeeded
```

Figure 16: GitLab CI setup Job Output

Automated documentation

Throughout this project, the generation of documentation is automated to ensure consistency and accuracy across all documentation as the project evolves. This documentation is ensured through the script `**generate_doc.sh**`. The script handles the complete setup, execution and generation of both HTML and PDF documentation using Sphinx, a documentation generator.

Sphinx uses docstrings that are written in "**reStructuredText**" format, as shown in Figure 17, and transforms it to a well formatted documentation. Displayed below is a sample of the docstring used to generate the documentation. This example is from the `**delete_file**` function from the test's module.

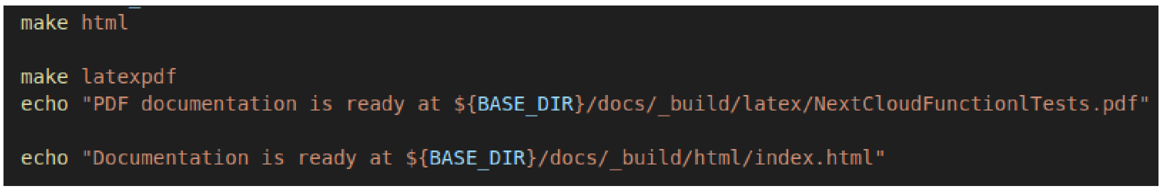


```
def delete_file(driver, file_name):
    """
    Deletes a specified file from NextCloud.

    This function navigates to the NextCloud files section by ensuring the session is logged in. It then attempts to
    delete a file with the given name. Successful deletion is verified internally by the function `perform_delete_file`.
    If the deletion process encounters any issues, an exception is raised.

    :param driver: An instance of Selenium WebDriver used for automating interactions with the NextCloud web
                   application. It should be properly initialized and configured prior to calling this function.
    :type driver: WebDriver
    :param file_name: The name of the file to be deleted from NextCloud. The name should match exactly with the file
                      present in NextCloud, including file extension.
    :type file_name: str

    :returns: True if the file is successfully deleted, False otherwise. This adjustment from the original docstring
              without a return statement provides clarity on the function's success through a boolean.
    :rtype: bool

    :raises AssertionError: If the file was not deleted successfully. This assertion could be raised within the
                            `perform_delete_file` function based on the verification of the file's absence after the
                            deletion attempt. Handling or logging this error in the calling context is recommended to
                            make informed workflow decisions based on the file deletion success.

    **Example**::

        >>> from selenium import webdriver
        >>> driver = webdriver.Chrome()
        >>> file_name = "example_document.pdf"
        >>> deletion_success = delete_file(driver, file_name)
```

Figure 17: Example of DocString for Documentation Generation

The `**generate_doc.sh**` script automates the setup and generation of both HTML and PDF documentation of the NextCloud functional tests. This script ensures that all documentation is consistently updated reflecting the latest changes in code. In this section we will explore the steps involved:

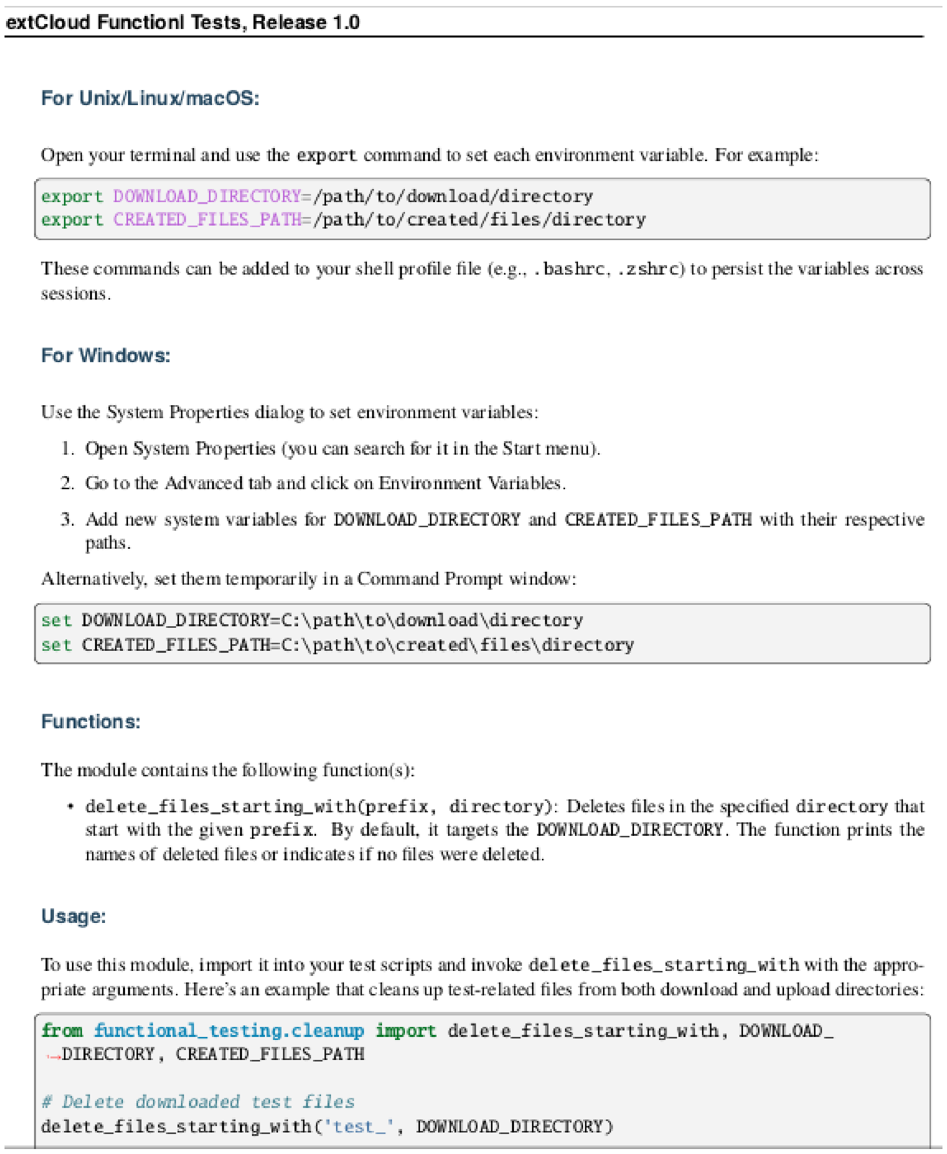
1. **Environment setup**: The script starts by determining and exporting the base directory to ensure that all next operation is relative to the root project.
2. **Dependency installation**: The script installs the necessary dependencies consisting of Sphinx and its themes, and LaTeX packages to enable the generation of PDFs.
3. **Project setup:** Ensures the python Library is installed and available to Sphinx.
4. **Documentation Initialization**: The script creates the directory that will host the generated documentation. It also runs `**sphinx-quickstart**` to set up a basic Sphinx configuration. Before generating the documentation files, the script ` **modify_conf.py**` is executed to alter the configuration. It ensures that the documentation aligns with the project's specific requirements.
5. **Documentation Build:** The final step is to generate source files from the Python source code using the command `**sphinx-apidoc –o docs .**` and build the HTML and PDF files using the command `**make html**` and `**make latexpdf**`.



```
make html

make latexpdf
echo "PDF documentation is ready at ${BASE_DIR}/docs/_build/latex/NextCloudFunctionlTests.pdf"

echo "Documentation is ready at ${BASE_DIR}/docs/_build/html/index.html"
```

Figure 18: Example of Automated Documentation Script

Figure 18 illustrates the process of generating the documentation as shown in Figure 19, which explains setting environment variables and utilizing functions within the NextCloud functional tests. As shown, it takes one command to generate the whole documentation.


extCloud Functionl Tests, Release 1.0

For Unix/Linux/macOS:

Open your terminal and use the `export` command to set each environment variable. For example:

```
export DOWNLOAD_DIRECTORY=/path/to/download/directory
export CREATED_FILES_PATH=/path/to/created/files/directory
```

These commands can be added to your shell profile file (e.g., `.bashrc`, `.zshrc`) to persist the variables across sessions.

For Windows:

Use the System Properties dialog to set environment variables:

1. Open System Properties (you can search for it in the Start menu).
2. Go to the Advanced tab and click on Environment Variables.
3. Add new system variables for `DOWNLOAD_DIRECTORY` and `CREATED_FILES_PATH` with their respective paths.

Alternatively, set them temporarily in a Command Prompt window:

```
set DOWNLOAD_DIRECTORY=C:\path\to\download\directory
set CREATED_FILES_PATH=C:\path\to\created\files\directory
```

Functions:

The module contains the following function(s):

- `delete_files_starting_with(prefix, directory)`: Deletes files in the specified `directory` that start with the given `prefix`. By default, it targets the `DOWNLOAD_DIRECTORY`. The function prints the names of deleted files or indicates if no files were deleted.

Usage:

To use this module, import it into your test scripts and invoke `delete_files_starting_with` with the appropriate arguments. Here's an example that cleans up test-related files from both download and upload directories:

```
from functional_testing.cleanup import delete_files_starting_with, DOWNLOAD_
→DIRECTORY, CREATED_FILES_PATH

# Delete downloaded test files
delete_files_starting_with('test_', DOWNLOAD_DIRECTORY)
```

Figure 19: Example of Generation Documentation for NextCloud Functional Tests

4.4. Iteration 3: Security and conformity auditing and hardening

4.4.1. Purpose of the iteration

This iteration is dedicated to ensuring security and integrity within the NextCloud infrastructure. It details how to incorporate conformance tests as well as the security auditing and hardening using advanced tools such as Sonobuoy and CIS benchmarks which ensures that the standards set for Ubuntu and Kubernetes are met. This phase not only addresses immediate security concerns but also sets the stage for continuous monitoring and validation, promising insights into maintaining a secure cloud-native application infrastructure.

4.4.2. Problem Investigation

The infrastructure on which the NextCloud application is deployed should ensure security, scalability, and integrity. We should ensure the conformity of the cluster to security benchmarks and best practices. For that reason, we need to define the frameworks and standards that we will be using. Specifically, the study will focus on CIS and CNCF benchmarks. Integration issues may arise due to the complexity of using security tools in a CI/CD pipeline. It is very important to identify these potential issues early and remediate them.

4.4.3. Solution Design:

Before diving into the solutions implemented to ensure the integrity, conformance and security of the infrastructure, this section starts with our approach to compliance. We opted for CIS benchmarks for both Ubuntu and Kubernetes. For conformance, Sonobuoy is implemented to ensure conformance to CNCF standards.

Other options include NIST SP 800-53 and ISO/IEC 27001, but we opted for CIS benchmarks with "kube-bench" and "USG" along with Sonobuoy for conformance since they are more specific to Cloud-Native environments, easier to integrate and have more community support and adoption.

Infrastructure Testing: Sonobuoy Conformance Testing

To ensure infrastructure integrity and compliance with Kubernetes standards and the Cloud Native Computing Foundation (CNCF), Sonobuoy was used. Sonobuoy is a diagnostic tool that helps understand the state of a Kubernetes cluster by running plugins in an accessible and non-destructive manner [20]. It is a customizable, extendable, and cluster-agnostic way to generate reports about the cluster. It offers workload debugging and custom data collection through plugins and integrated end-to-end conformance testing for Kubernetes [20]. For this project we will perform the latter tests, to ensure that the configuration is correct, and that the behavior follows the best practices [18].

The objectives of these tests are:

- **Integrity**: To verify that the Kubernetes cluster configuration is correct and follows best practices.
- **Compliance:** Ensure the cluster conforms to CNCF standards.
- **Reporting:** To generate clear detailed reports on the cluster state.

When automated and integrated to the CI/CD pipeline the tests are executed after each cluster deployment. This ensures continuous compliance and integrity of the infrastructure [18]. Steps to integrate these tests to the pipeline generally follow the same steps:

1. **Cluster connection**: To establish a connection to the Kubernetes cluster.
2. **Environment preparation**: Download and configure Sonobuoy.
3. **Test Execution**: Run Sonobuoy tests.
4. **Result retrieval**: Retrieve and analyze the test results.
   Reporting: Generate and store reports.

Incorporating Sonobuoy for testing the infrastructure provides a strong automated solution to ensure the integrity and compliance of the NextCloud Kubernetes cluster. By exploiting its diagnostic and compliance capabilities, performance and health of the cluster can be maintained.

Security Testing with Benchmark Compliance

To ensure the security of the NextCloud infrastructure, adherence to the Center for Internet Security (CIS) benchmarks is one of the most effective solutions. They are internationally recognized for defending IT systems. They include the configuration of operating systems and applications. For this project, hardening the Ubuntu virtual machines and the Kubernetes cluster is essential to achieve compliance with the previously mentioned standards.

Hardening Ubuntu Virtual Machines

The ubuntu Security Guide (USG) is a very important tool for compliance and auditing. It simplifies the process of hardening Ubuntu systems, and it is part of Ubuntu Advantage and Ubuntu Pro. It allows for the application of CIS compliance rules through simple commands.

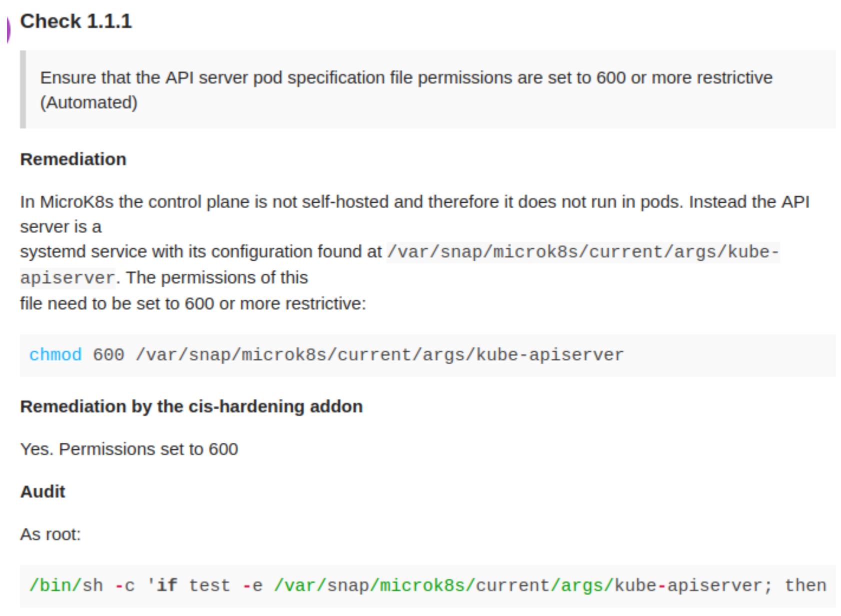
For example, running `**sudo usg fix <PROFILE>**` modifies the system to comply with the specified CIS profile.

The tools also provide auditing capabilities, using the `**usg audit <PROFILE>**` command. This generates an HTML report detailing the compliance status. It includes the tests that passed, warnings and critical issues to provide a clear overview of the system's security. The USG tool also allows customization of the CIS benchmarks profiles to tailor the compliance checks for a specific use case and environment need.

Hardening the Kubernetes Cluster

The security of the Kubernetes cluster is also evaluated using `**kube-bench**`, a tool that checks the cluster against the CIS Kubernetes Benchmarks. Same as those of Ubuntu, they include recommendations for securing cluster components. `**kube-bench**` performs automated checks and produces detailed reports to assess components such as the API server, controller manager, etcd, kubelet and kube-proxy, and categorizes each test to 'Pass' or 'Fail'.

Running `**kube-bench**` applies a Kubernetes job that executes the benchmark tests in the cluster. The results are then collected and analyzed to identify the failures and their severity and try to remediate them. Since each failed test comes with its remediations in the official documentation of Kubernetes as shown in Figure 20.



Check 1.1.1

> Ensure that the API server pod specification file permissions are set to 600 or more restrictive (Automated)

Remediation

In MicroK8s the control plane is not self-hosted and therefore it does not run in pods. Instead the API server is a
systemd service with its configuration found at `/var/snap/microk8s/current/args/kube-apiserver`. The permissions of this
file need to be set to 600 or more restrictive:

```
chmod 600 /var/snap/microk8s/current/args/kube-apiserver
```

Remediation by the cis-hardening addon

Yes. Permissions set to 600

Audit

As root:

```
/bin/sh -c 'if test -e /var/snap/microk8s/current/args/kube-apiserver; then
```

Figure 20: CIS Benchmark Check Example

CIS compliance checks and remediations will be automated and integrated within the CI pipeline to ensure continuous monitoring and enforcement of security standards.

4.4.4. Design Validation

In this section we will evaluate the effectiveness of the proposed solution and ensure they solve the stated problem.

Internal Validity

Internal Validity ensures that the results of the security and conformance testing and auditing are attributable to the implemented hardening. It also makes sure that no external factors can impact these results. By performing audits before and after the hardening and remediations steps, we ensure that improvements are related directly to the modifications made.

Trade-offs

The most important trade-off in this iteration is balancing security measures and potential impact on performance and deployment time. It necessitates careful planning and optimization of the CI/CD pipeline to shorten the execution time without affecting security.

External Validity

The proposed solution is applicable and generalizable beyond the context of NextCloud. Standards such as CIS and CNCF are using in a very wide range used in a very wide range of different platforms and applications and can be generalized easily.

4.4.5. Solution implementation

The solution implementation for ensuring the integrity, compliance, and security of the NextCloud application involves a series of steps and stages, automated through the CI/CD pipeline using GitLab. This section details the process, including the scripts and commands used to achieve the desired outcomes.

CI/CD Pipeline Overview

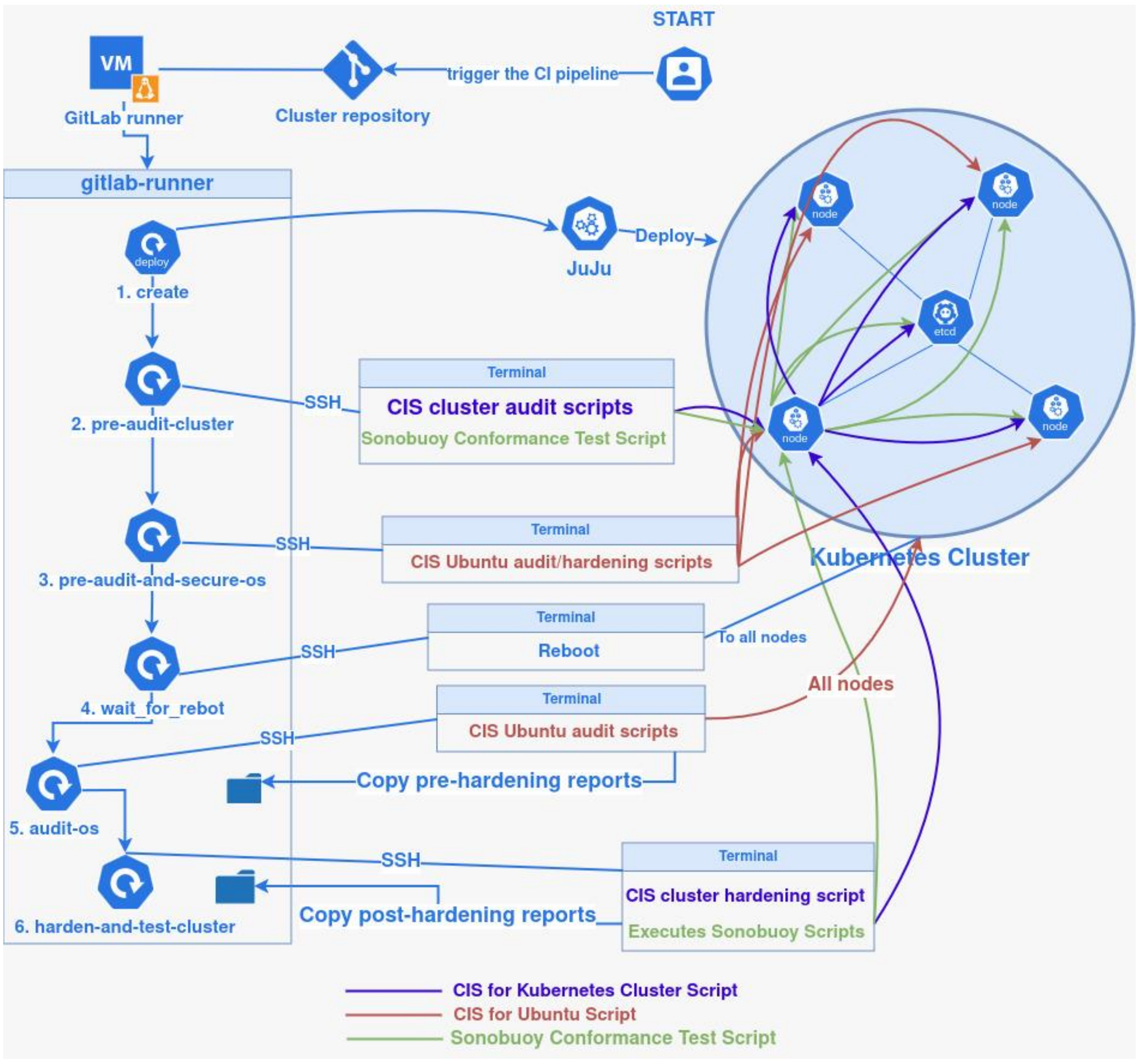

START
VM
GitLab runner
Cluster repository
trigger the CI pipeline
gitlab-runner
deploy
1. create
JuJu
Deploy
2. pre-audit-cluster
SSH
Terminal
CIS cluster audit scripts
Sonobuoy Conformance Test Script
3. pre-audit-and-secure-os
SSH
Terminal
CIS Ubuntu audit/hardening scripts
4. wait_for_rebot
SSH
Terminal
Reboot
To all nodes
5. audit-os
SSH
Terminal
CIS Ubuntu audit scripts
All nodes
Copy pre-hardening reports
6. harden-and-test-cluster
SSH
Terminal
CIS cluster hardening script
Executes Sonobuoy Scripts
Copy post-hardening reports
Kubernetes Cluster
node
etcd
CIS for Kubernetes Cluster Script
CIS for Ubuntu Script
Sonobuoy Conformance Test Script

Figure 21: CI/CD Pipeline Workflow for Kubernetes Cluster Deployment and Testing

As shown in the Figure 21, the CI/CD pipeline is divided into several stages, each performing specific tasks to ensure the system's security, compliance, and functionality. The stages are:

1. **Create stage**: It deploys the cluster and its components.
2. **Pre-audit-cluster stage:** It performs audits against the CIS and CNCF benchmarks before performing any hardening.
3. **Secure-and-audit-os stage**: Hardening the operating systems and providing reports of the before and after state of the machines. Also copies all reports to the gitlab-runner node.
4. **Wait and reboot**: Ensuring all systems are rebooted after the secure stage so that remediations are applied.
5. **Test stage**: Hardens the Kubernetes cluster against CIS benchmarks and performs security audits and copies all reports to the gitlab-runner.

The CI/CD pipeline configuration is defined in the GitLab CI file as follows:

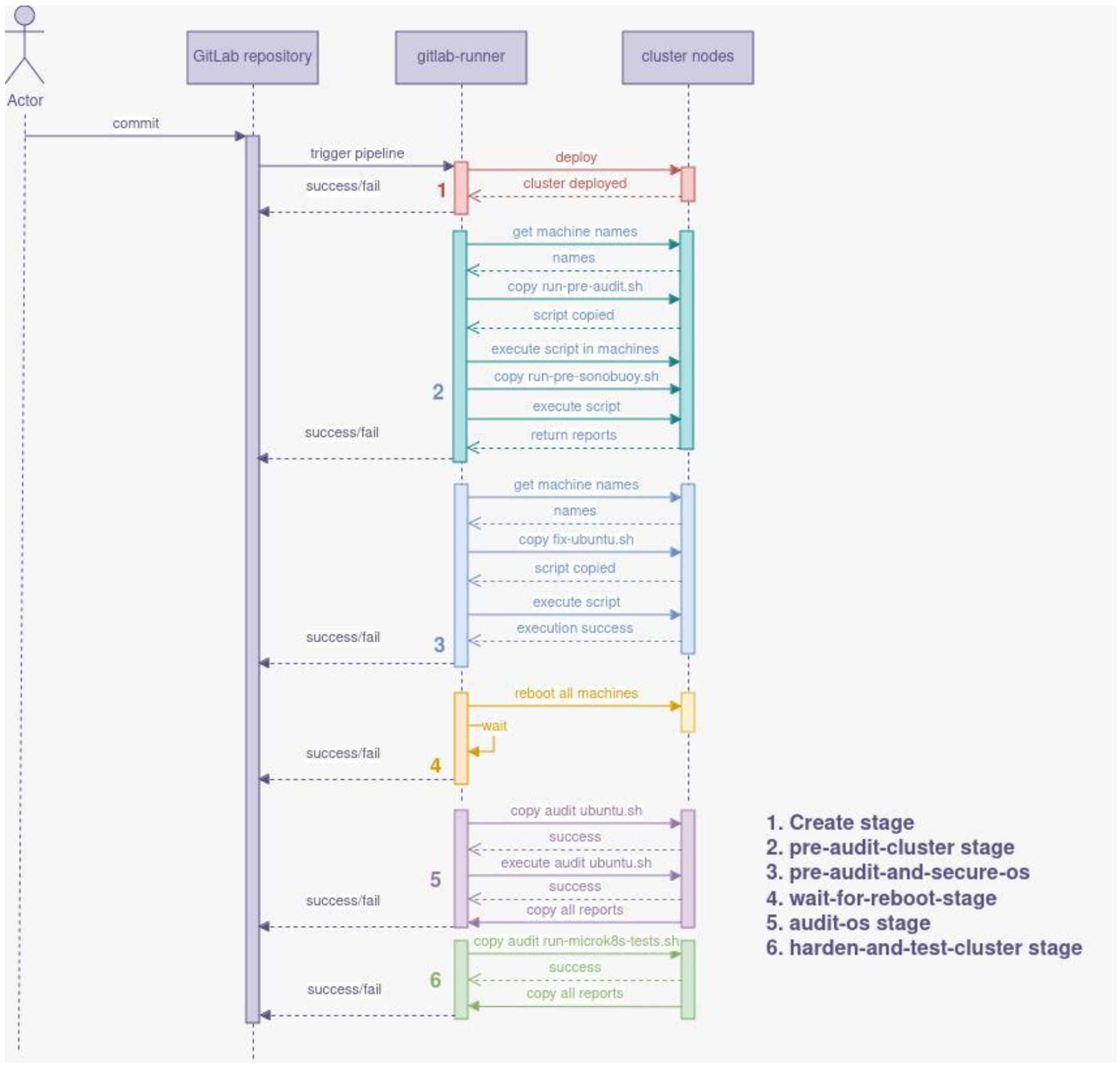


Figure 22: Sequence diagram of the CI/CD Pipeline

As explained in Figure 21 and Figure 22, the process starts when a commit is made to the GitLab repository, which triggers the CI/CD pipeline. The pipeline is divided into several stages: `Secure`, `Wait for Reboot`, `Audit` and `Test Cluster`. Each stage has specific steps executed to ensure the deployment, security, and performance of the infrastructure.

Triggering the pipeline: A commit to the GitLab repository triggers the CI/CD pipeline and the Kubernetes cluster is deployed.

Secure stage: The script communicates with Juju API to retrieve the names of the machines in the cluster as displayed in the figure below. Then it copies `fix-ubuntu.sh` to all machines in the cluster. The pipeline executes the fix scripts in all Ubuntu virtual machines in the cluster iteratively, to apply CIS standards. The script, shown below, uses Ubuntu Security Guide package to generate and execute a set of predefined rules, to harden the machines.

Wait for Reboot Stage: After the execution of the fixes, the machines are rebooted to reflect the new configurations. The pipeline sends a reboot message to all machines and halts for 10min.

Audit Stage: The Audit stage is a very important part of the CI/CD pipeline since it assures infrastructure security and compliance. It is configured to run deep audits regarding security in each of the machines that are part of the Kubernetes cluster in compliance with the CIS benchmarks. This process involves several automated steps, orchestrated by a CI/CD pipeline directly into GitLab. The implementation is shown in the figure below.

As captured in **Error! Reference source not found.**, the Audit stage starts with the pipeline switching to the correct Juju model in which the Kubernetes cluster was deployed. This is an important step as any further commands would need to be executed in the right environment. The pipeline then gets a list of all machine names within the Kubernetes cluster by querying the Juju API for that information. Getting the names of the machines would be an operation based on iteration; that is, an operation that would be done for each machine to carry out activities for the audit.

The figure below illustrates the audit process. From the start to the creation of the artifacts.

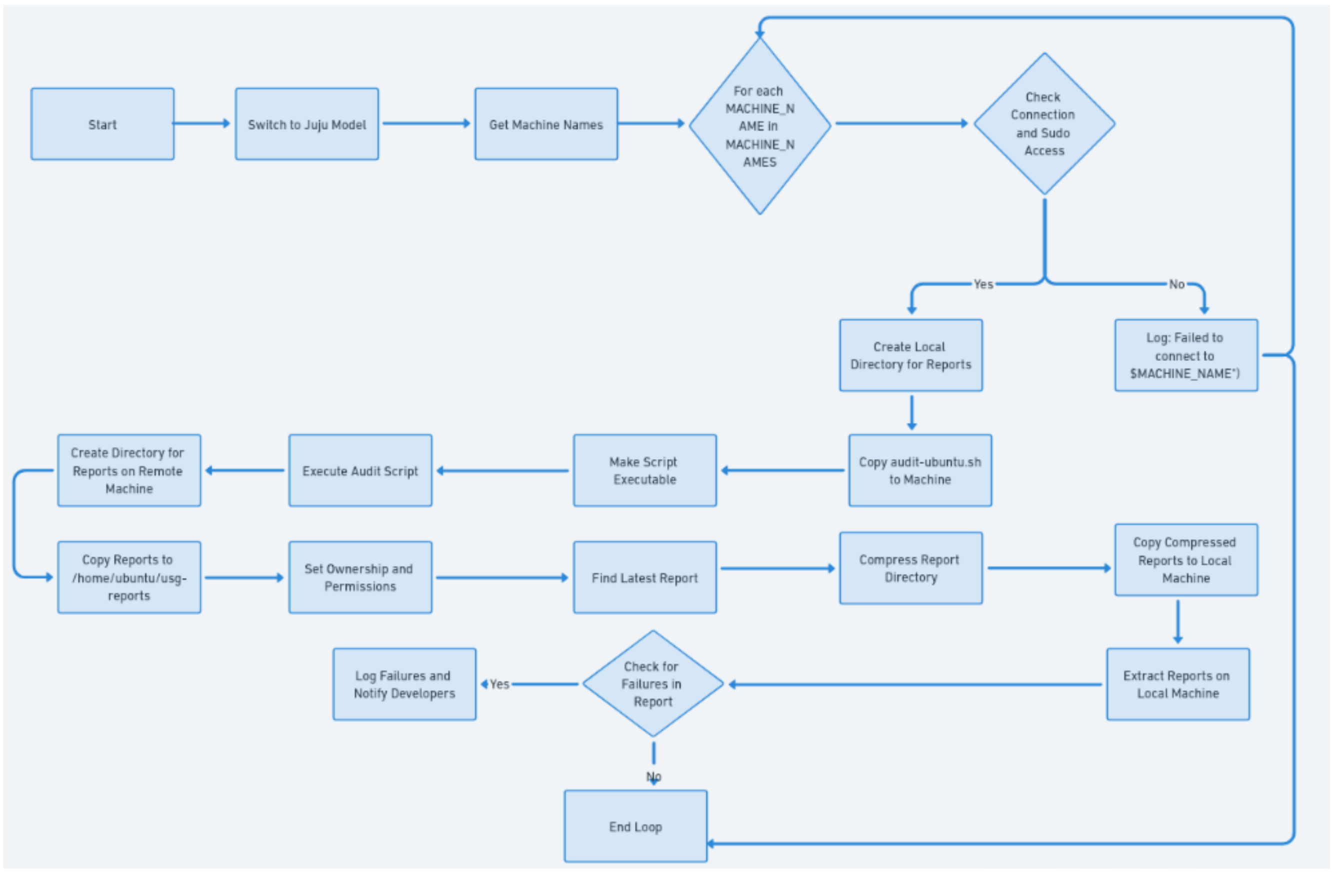
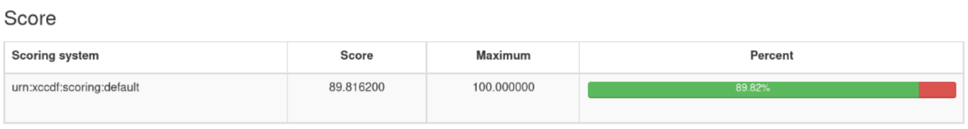

Start
Switch to Juju Model
Get Machine Names
For each MACHINE_NAME in MACHINE_NAMES
Check Connection and Sudo Access
Yes
No
Create Local Directory for Reports
Log: Failed to connect to $MACHINE_NAME")
Copy audit-ubuntu.sh to Machine
Make Script Executable
Execute Audit Script
Create Directory for Reports on Remote Machine
Copy Reports to /home/ubuntu/usg-reports
Set Ownership and Permissions
Find Latest Report
Compress Report Directory
Copy Compressed Reports to Local Machine
Extract Reports on Local Machine
Check for Failures in Report
Yes
Log Failures and Notify Developers
No
End Loop

Figure 23: Workflow for Executing and Retrieving Audit Reports

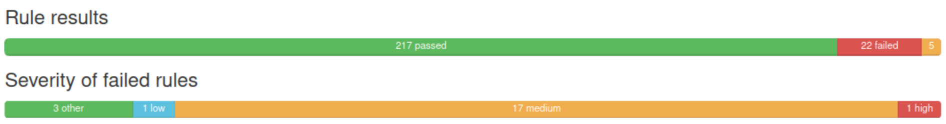
The next step is to copy the script `**audit-ubuntu.sh**` to all the machines in the cluster. This script performs a complete security audit with reference to the CIS benchmarks. It runs different commands and checks to find the state of security of the machine and provides detailed reports in HTML format as shown in Figure 24, **Error! Reference source not found.** and Figure 26.



Score

Scoring system	Score	Maximum	Percent
urn:xccdf:scoring:default	89.816200	100.000000	89.82%

Figure 24: Compliance Scoring System for CIS Benchmarks



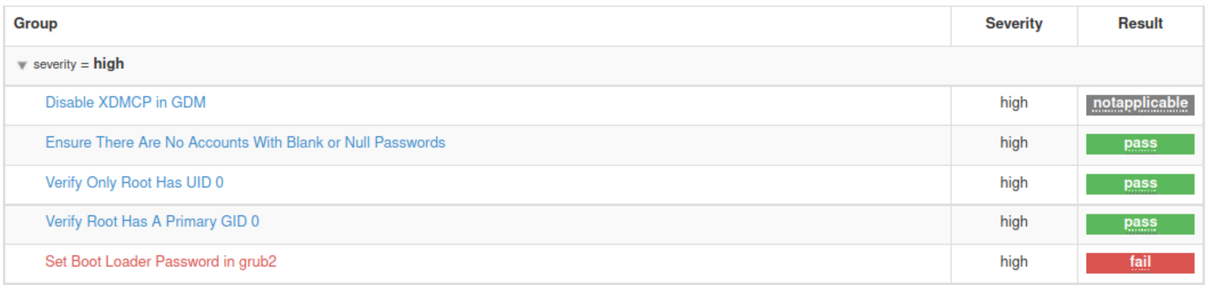
Rule results

217 passed | 22 failed | 5

Severity of failed rules

3 other | 1 low | 17 medium | 1 high

Figure 25: Rule Results and Severity of Failed Rules

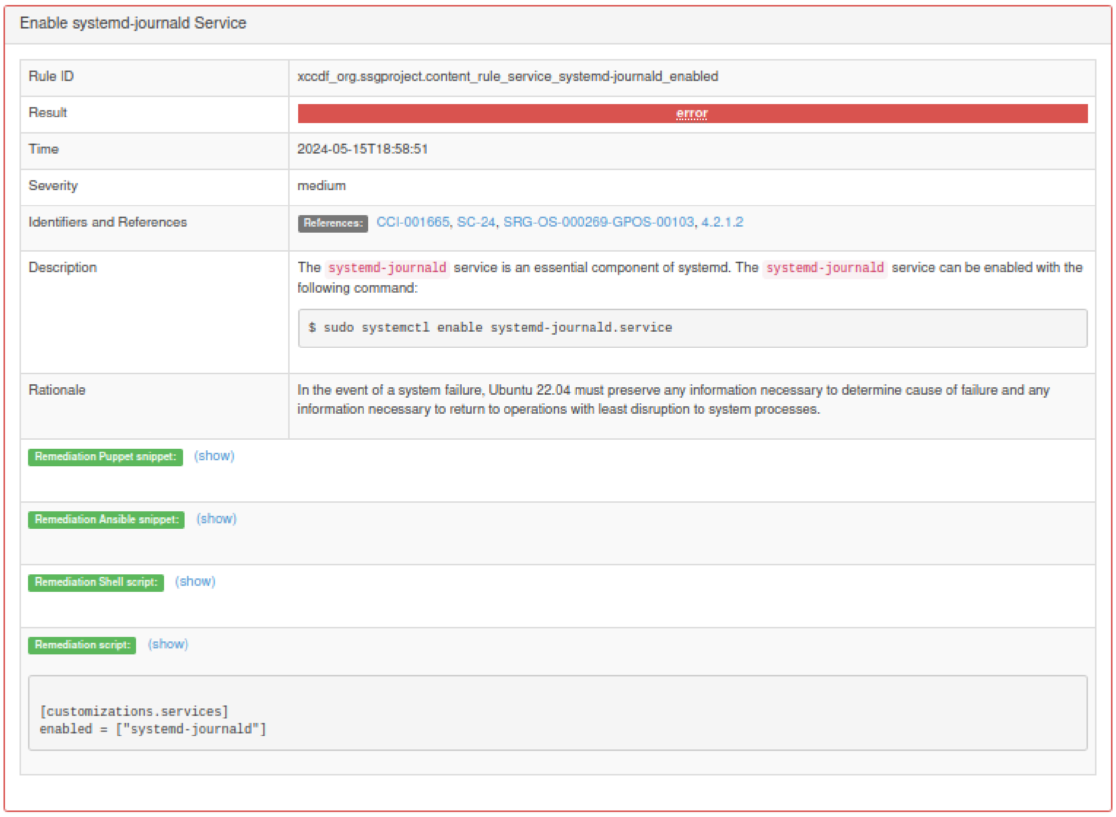


Group	Severity	Result
severity = **high**		
Disable XDMCP in GDM	high	notapplicable
Ensure There Are No Accounts With Blank or Null Passwords	high	pass
Verify Only Root Has UID 0	high	pass
Verify Root Has A Primary GID 0	high	pass
Set Boot Loader Password in grub2	high	fail

Figure 26: High Severity Compliance Results

After the audit script has been executed, the pipeline copies the generated reports back to the machine that hosts the GitLab runner. The reports are then scanned for signs of failed tests or errors. Such processes, where one is pre-informed, assist the development team in immediately taking any security issue into action, hence making the infrastructure compliant and intact. Essentially, the CI/CD pipeline has been designed with the Audit phase in place to fully automate auditing of security within our Ubuntu Virtual machines. This should ensure that machines are checked against CIS benchmarks, reports are created, and easily transferred for analysis. Integration into the CI/CD pipeline gives a good mechanism to ensure security and compliance. It also provides the teams with detailed reports.

Considering the report, along with the scoring and compliance, it provides a Rule Results section, which analyzes the compliance status by specifying the severity of failed rules. It divides the failures into low, medium, and high severity as shown in Figure 24. This enables the development team to set priorities for remediation actions with respect to the potential impact of non-compliance. For example, as shown in Figure 27, the rule indicates that system services such as `systemd-journald`, `rsyslog`, and `cron` should be enabled. It also details the remediation steps to correct the issue as shown in both Figure 27 and Figure 28.



Enable systemd-journald Service

Rule ID	xccdf_org.ssgproject.content_rule_service_systemd-journald_enabled
Result	error
Time	2024-05-15T18:58:51
Severity	medium
Identifiers and References	References: CCI-001665, SC-24, SRG-OS-000269-GPOS-00103, 4.2.1.2
Description	The `systemd-journald` service is an essential component of systemd. The `systemd-journald` service can be enabled with the following command: `$ sudo systemctl enable systemd-journald.service`
Rationale	In the event of a system failure, Ubuntu 22.04 must preserve any information necessary to determine cause of failure and any information necessary to return to operations with least disruption to system processes.

Remediation Puppet snippet: (show)

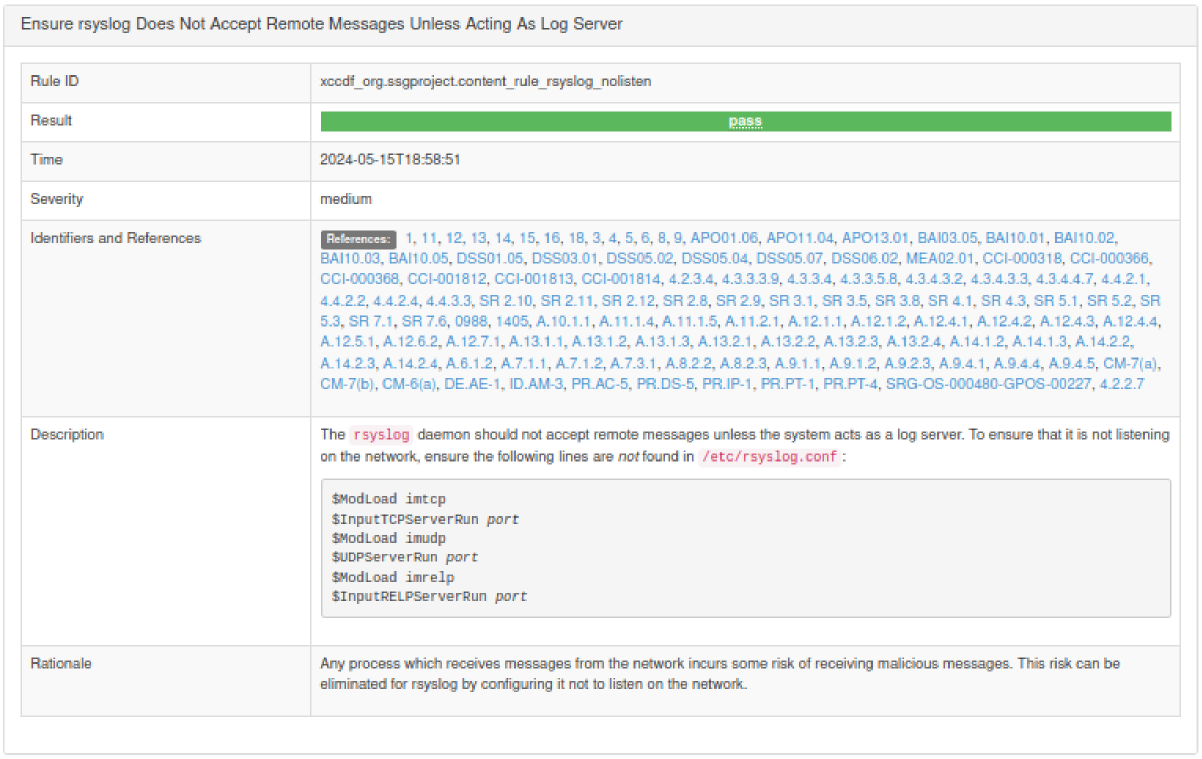
Remediation Ansible snippet: (show)

Remediation Shell script: (show)

Remediation script: (show)

```
[customizations.services]
enabled = ["systemd-journald"]
```

Figure 27: Enable system-journalld Service Compliance Result



Ensure rsyslog Does Not Accept Remote Messages Unless Acting As Log Server

Rule ID	xccdf_org.ssgproject.content_rule_rsyslog_nolisten
Result	pass
Time	2024-05-15T18:58:51
Severity	medium
Identifiers and References	References: 1, 11, 12, 13, 14, 15, 16, 18, 3, 4, 5, 6, 8, 9, APO01.06, APO11.04, APO13.01, BAI03.05, BAI10.01, BAI10.02, BAI10.03, BAI10.05, DSS01.05, DSS03.01, DSS05.02, DSS05.04, DSS05.07, DSS06.02, MEA02.01, CCI-000318, CCI-000366, CCI-000368, CCI-001812, CCI-001813, CCI-001814, 4.2.3.4, 4.3.3.3.9, 4.3.3.4, 4.3.3.5.8, 4.3.4.3.2, 4.3.4.3.3, 4.3.4.4.7, 4.4.2.1, 4.4.2.2, 4.4.2.4, 4.4.3.3, SR 2.10, SR 2.11, SR 2.12, SR 2.8, SR 2.9, SR 3.1, SR 3.5, SR 3.8, SR 4.1, SR 4.3, SR 5.1, SR 5.2, SR 5.3, SR 7.1, SR 7.6, 0988, 1405, A.10.1.1, A.11.1.4, A.11.1.5, A.11.2.1, A.12.1.1, A.12.1.2, A.12.4.1, A.12.4.2, A.12.4.3, A.12.4.4, A.12.5.1, A.12.6.2, A.12.7.1, A.13.1.1, A.13.1.2, A.13.1.3, A.13.2.1, A.13.2.2, A.13.2.3, A.13.2.4, A.14.1.2, A.14.1.3, A.14.2.2, A.14.2.3, A.14.2.4, A.6.1.2, A.7.1.1, A.7.1.2, A.7.3.1, A.8.2.2, A.8.2.3, A.9.1.1, A.9.1.2, A.9.2.3, A.9.4.1, A.9.4.4, A.9.4.5, CM-7(a), CM-7(b), CM-6(a), DE.AE-1, ID.AM-3, PR.AC-5, PR.DS-5, PR.IP-1, PR.PT-1, PR.PT-4, SRG-OS-000480-GPOS-00227, 4.2.2.7

Description	The `rsyslog` daemon should not accept remote messages unless the system acts as a log server. To ensure that it is not listening on the network, ensure the following lines are *not* found in `/etc/rsyslog.conf`:
	`$ModLoad imtcp`
	`$InputTCPServerRun port`
	`$ModLoad imudp`
	`$UDPServerRun port`
	`$ModLoad imrelp`
	`$InputRELPServerRun port`

Rationale	Any process which receives messages from the network incurs some risk of receiving malicious messages. This risk can be eliminated for rsyslog by configuring it not to listen on the network.

Figure 28: Ensure rsyslog Does Not Accept Remote Messages Compliance Result

Test Cluster Stage: The Test Cluster Stage is importance in the CI/CD pipeline and basically guarantees that our Kubernetes cluster is compliant with the chosen security benchmarks. Cluster validation is done with Sonobuoy and CIS tests under the CNCF and CIS standards. It starts by checking out the right Juju model, where all the commands are run in the right context. This is an important step, as it gives context to all future steps, allowing the pipeline to correctly interact with the cluster.

Next, the pipeline checks if the model received the appropriate permissions to the main node of a Kubernetes cluster by executing a very simple `whoami` command and copies the **run-microk8s-tests.sh** script to the main node. The script will first check the readiness of MicroK8s, to ensure that the Kubernetes cluster is up and stable before running any tests. It enforces CIS hardening on MicroK8s using predefined security configurations. It is set to run `**kube-bench**`, which is one of the tools to check the security of Kubernetes clusters against the CIS benchmarks. The tool will execute testing rules to ensure there are no deviations from the set benchmarks, indicating areas that need correction. Additionally, conformance tests are conducted using Sonobuoy. **Error! Reference source not found.** depicts downloading and preparing the Kubernetes diagnostic tool, Sonobuoy, to run a suite of conformance and performance tests.

After the completion of the Sonobuoy tests, the script goes ahead to fetch the results and extract them into a provided directory. These results are informative to the cluster level of compliance and performance on a much deeper level, calling out areas requiring remediation. That just means the Kubernetes cluster is validated, at this step, against CNCF and CIS benchmarks and validated for production readiness. The reports, at the granular level, established with this phase, help to maintain the security and compliance of the cluster. They give a clear picture of the present state of the cluster, identifying any security gaps or configuration issues. This data is valuable to the development team, who will then know how to cope with the issues in time to keep the Kubernetes environment secure and compliant. The inclusion of such tests in the CI/CD pipeline provides continuous monitoring and validation, giving a robust mechanism for the maintenance of the Kubernetes cluster's integrity.

5. RESULTS AND FINDINGS

5.1. Answer to the first Research Question

The first research question was divided into three sub research questions to be able to answer more precisely and specifically to each aspect.

5.1.1. Answer to the first Sub-Question

For answering the 1st sub-question "**What measurable improvements in quality and security are observed following the integration of automated testing into NextCloud's CI/CD pipeline**?", we will study specific improvements at the virtual machine level and within the Kubernetes Cluster.

Improvements at the virtual machine level

Regarding the operating systems installed in the virtual machines assigned to the Kubernetes cluster, the study offered important Insights on the potential benefits as illustrated by Figure 29. The automated tests integrated into the CI/CD flow of NextCloud have delivered high and measurable improvement on security metrics. The first audit based on the CIS benchmarks, conducted before the hardening steps, reported a high number of vulnerabilities and non-compliance in the tested ubuntu operated system. As showed in Figure 30 the audit report identified 109 failures of 228 points, leading to a compliance score of 64.08%. Examples from the highly critical areas of failure include disabled critical security services like `systemd-journald` and `rsyslog`, not properly configuring user authentication settings; and not implementing network security parameters.

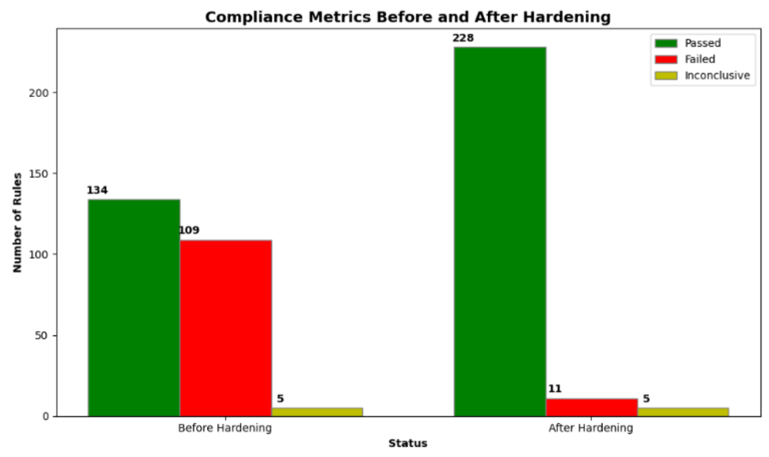
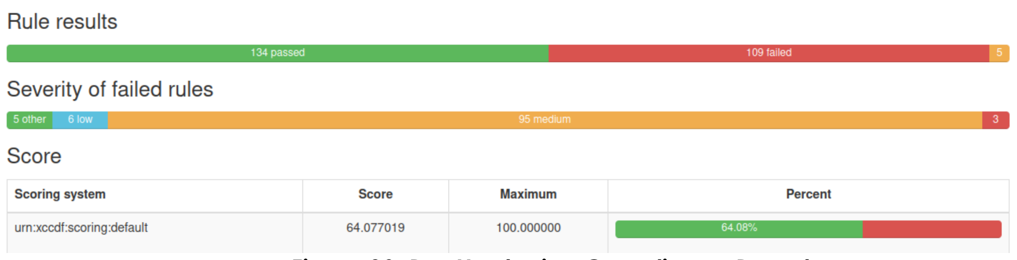


Figure 29: Compliance Metrics Before and After Hardening of Ubuntu VM



Rule results

134 passed | 109 failed | 5

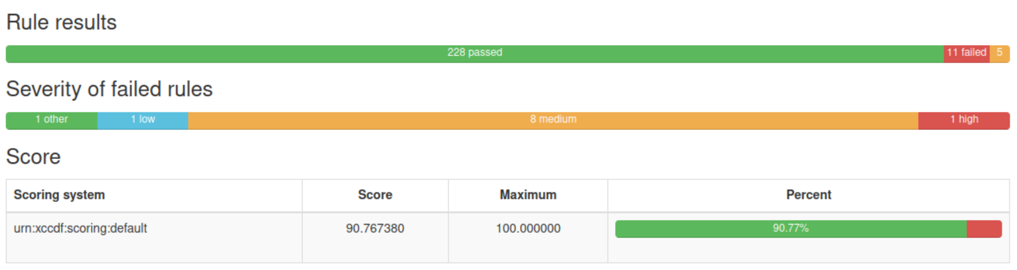
Severity of failed rules

5 other | 6 low | 95 medium | 3

Score

Scoring system	Score	Maximum	Percent
urn:xccdf:scoring:default	64.077019	100.000000	64.08%

Figure 30: Pre-Hardening Compliance Report



Rule results

228 passed | 11 failed | 5

Severity of failed rules

1 other | 1 low | 8 medium | 1 high

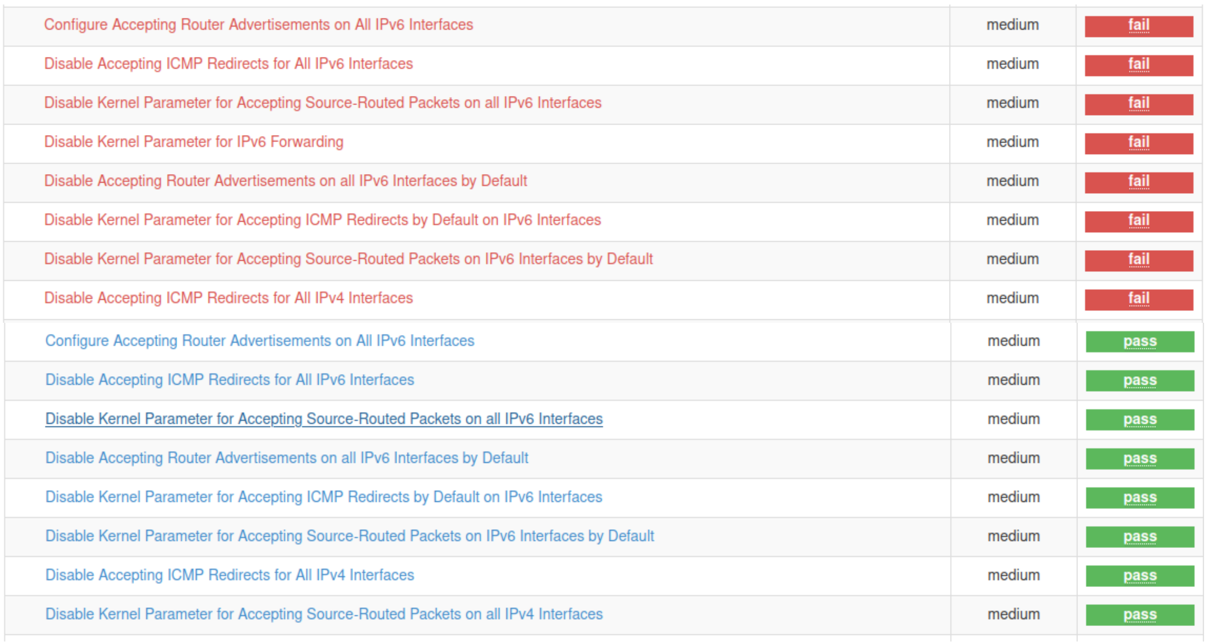
Score

Scoring system	Score	Maximum	Percent
urn:xccdf:scoring:default	90.767380	100.000000	90.77%

Figure 31: Post-Hardening Compliance Report

The results were found to be highly improved after the execution of security-hardening measures. As shown in Figure 31 and Figure 29 the failed test count severely was down to 11 out of 244, and the compliance score improved to 90.77%. The results prove the effectiveness of automated testing and hardening steps built within the CI/CD pipeline. It guaranteed that the important security services were on, that proper user authentication mechanisms were configured, and that the settings related to the network security were tuned according to the CIS benchmarks.

To illustrate, the pre-hardening audit had listed services for enabling `systemd-journald` and `rsyslog`, that provide important aspects of logging and monitoring system activity. After hardening, services have been enabled, as listed by resolved audit checks. A first set of failures listed regarding user authentication configurations—that is, boot loader passwords and the need to re-authenticate for `sudo` commands—was resolved by means of an automation script that enforced these configurations across all nodes within the Kubernetes cluster. Compliance with network security settings was also enhanced by the hardening process. For instance, before the intervention, many kernel parameters concerning the forwarding of IPv4, IPv6 and the accepting of ICMP redirects were in the wrong settings. As mentioned in the figures Figure 30, Figure 31 and Figure 32 which compare both states before and after hardening the system. After automated fixes, these parameters were properly configured, increasing system network security stance.



Configure Accepting Router Advertisements on All IPv6 Interfaces	medium	fail
Disable Accepting ICMP Redirects for All IPv6 Interfaces	medium	fail
Disable Kernel Parameter for Accepting Source-Routed Packets on all IPv6 Interfaces	medium	fail
Disable Kernel Parameter for IPv6 Forwarding	medium	fail
Disable Accepting Router Advertisements on all IPv6 Interfaces by Default	medium	fail
Disable Kernel Parameter for Accepting ICMP Redirects by Default on IPv6 Interfaces	medium	fail
Disable Kernel Parameter for Accepting Source-Routed Packets on IPv6 Interfaces by Default	medium	fail
Disable Accepting ICMP Redirects for All IPv4 Interfaces	medium	fail
Configure Accepting Router Advertisements on All IPv6 Interfaces	medium	pass
Disable Accepting ICMP Redirects for All IPv6 Interfaces	medium	pass
Disable Kernel Parameter for Accepting Source-Routed Packets on all IPv6 Interfaces	medium	pass
Disable Accepting Router Advertisements on all IPv6 Interfaces by Default	medium	pass
Disable Kernel Parameter for Accepting ICMP Redirects by Default on IPv6 Interfaces	medium	pass
Disable Kernel Parameter for Accepting Source-Routed Packets on IPv6 Interfaces by Default	medium	pass
Disable Accepting ICMP Redirects for All IPv4 Interfaces	medium	pass
Disable Kernel Parameter for Accepting Source-Routed Packets on all IPv4 Interfaces	medium	pass

Figure 32: Comparison of Compliance Results Before and After Hardening

To summarize, the automation of security testing and hardening of the operating system that is integrated with CI/CD have achieved proven enhancements in compliance scores and reduction of security vulnerabilities.

Kubernetes Cluster

The pre-hardening audit of the Kubernetes cluster, executed with the kube-bench tool, revealed several critical vulnerabilities and non-compliance issues. These issues primarily included inadequate permissions and ownership settings for configuration files, insufficient TLS configurations, and improper authorization and authentication mechanisms. For example, permissions for key files such as `admin.conf`, `scheduler.conf`, and `controller-manager.conf` were not set to the recommended levels, and several API server parameters like `--authorization-mode`, `--secure-port`, and `--audit-log-path` were either incorrectly configured or missing. This initial audit showed 17 checks passed, 53 checks failed, and 54 checks had warnings.

The post-hardening audit report showed a decrease in the number of failed checks, from 53 down to 0, and the number of warnings reduced from 54 to 41. The number of checks that passed increased dramatically from 17 to 83. Key improvements were observed in several critical areas. For example, the API server configuration was enhanced to include proper authorization modes and secure port settings, while the necessary TLS parameters were correctly configured. File permissions and ownership for critical Kubernetes configuration files were also set according to best practices, ensuring better security controls over the cluster's sensitive data.

The bar charts in Figure 33 compares the number of passed, failed, and warning checks before and after hardening, providing a clear visual representation of the improvements.

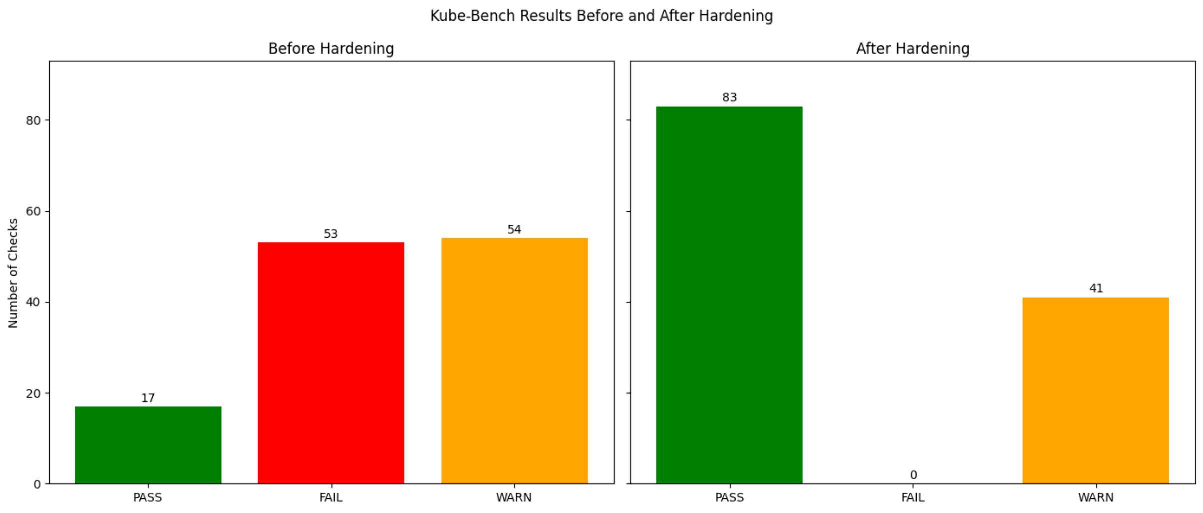


Figure 33: Kube-Bench Results Before and After CIS Hardening

In summary, the automation of these security hardening steps within the CI/CD pipeline has proven to be effective in enhancing the security posture of the Kubernetes cluster and Ubuntu virtual machines. This process not only ensures continuous compliance with CIS benchmarks but also significantly reduces the risk of vulnerabilities being introduced into the cluster. The findings highlight the importance of automated security tools and processes in maintaining the security and integrity of cloud-native applications. Through continuous monitoring and automated remediation, organizations can achieve a robust security baseline, ensuring their Kubernetes environments are secure and resilient against potential threats.

5.1.2. Answer to the second Sub-Question

Complexity of Setup and Configuration

For the 2nd sub-question "What challenges are associated with integrating these automated tests into the CI/CD pipeline, and how can they be mitigated to optimize NextCloud's deployment?", the automated testing in the NextCloud CI/CD pipeline faced some challenges in integration, configuration, and setup. This part of the process should be fully automated, requiring the `gitlab-runner` to have comprehensive permissions to copy scripts, execute them, and store the reports in a completely automated manner. It is very important to consider all machines involved to ensure seamless automation across the entire infrastructure. Additionally, integrating functional tests that simulate user interactions is particularly challenging when running in a headless mode. These tests often encounter issues related to UI interactions, making it difficult to pinpoint the cause of failures. Simulating the UI and running tests in headless mode further complicates the process. In such cases, detailed documentation is crucial. Tools for auto-generating documentation, such as Sphinx, help maintain consistency and clarity. Employing configuration management tools like Ansible can automate the setup process and standardize configurations across different environments, thereby reducing complexity.

Time to Pipeline Execution and Existing Workflow Integration

Comprehensive testing, such as Sonobuoy and CIS benchmarks, increase pipeline execution time. The process initially took about 20 minutes and after implementation of the scripts is extended to several hours. Functional tests also double the deployment time of the NextCloud application. Furthermore, any test failure results in the entire pipeline failing, halting the deployment process. This makes the integration of tests highly sensitive and requires careful consideration. To address this issue, workflows must be refactored to include stages for automated testing. Collaboration with development and operations teams is essential to ensure seamless integration without disrupting existing processes. By structuring the workflow to accommodate testing stages, the overall impact on deployment time can be managed more effectively.

To summarize, integrating automated tests into the NextCloud CI/CD pipeline presents challenges such as setup complexity, increased pipeline execution time, and issues with headless UI tests. These challenges can be mitigated by automating permissions and configurations using tools like Ansible, maintaining detailed documentation with tools like Sphinx, and refactoring workflows to include dedicated testing stages. This approach ensures seamless integration and efficient management of deployment time.

5.1.3. Answer to the third Sub-Question

For the second sub-question, "**What are the primary benefits realized from integrating automated security and performance testing into the CI pipeline for NextCloud, and how do they balance against the challenges?",** we will examine the complexity of setup and configuration, as well as the time required for pipeline execution and the integration of automated tests.

Improved Security Posture

Iintegrating automated security testing into the CI/CD pipeline of NextCloud has significantly enhanced its security posture. Regularly running CIS benchmarks ensures continuous monitoring and enforcement of security standards. This proactive approach helps identify vulnerabilities at earlier stages, allowing for immediate remediation before they can be exploited. For instance, automated security tests have detected configuration issues and potential security gaps, enabling the team to address these vulnerabilities, which brings us to the next point.

Early Detection of Issues

Automated testing facilitates the early detection of issues in the development cycle, minimizing the impact functionalities not working as expected and problems with the configurations of the deployment. By running automated tests at various stages of the CI pipeline, potential problems are caught early, preventing them from progressing to later stages where they could cause more significant disruptions. For example, early detection of security vulnerabilities ensures that these issues are resolved before they affect the end users. This early intervention is crucial in maintaining the stability and reliability of the NextCloud application.

Increased Confidence in Core Functionalities

The implementation of automated functional tests that test the core functionalities of NextCloud has increased confidence in the system's reliability. These tests simulate user interactions and validate that critical features work as intended. This increased confidence is particularly important during deployment, as it reduces the risk of introducing defects into the production environment.

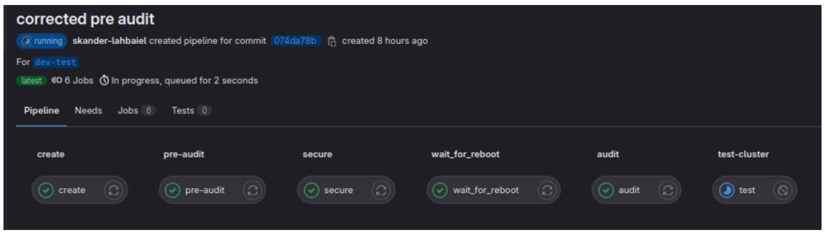
Enhanced Compliance

Automated testing ensures adherence to industry standards and regulatory requirements, enhancing compliance across the NextCloud deployment. Continuous compliance checks simplify audits and ensure that the application remains compliant over time. By integrating compliance tests into the CI/CD pipeline, the development team can quickly identify and address non-compliance issues, maintaining a high standard of governance. This proactive approach to compliance reduces the risk of regulatory penalties and builds trust with users and stakeholders.

5.2. Answer to the second Research Question

When answering the research question "**How can CIS hardening be effectively integrated into the development and deployment process of containerized applications to enhance security without negatively impacting key metrics such as deployment time and resource usage?**", the strategic approach in adopting CIS hardening at the build and release life cycle of NextCloud has proved to guarantee a better security posture without key performance indicator trade-offs relating to deployment time and resource usage. This can be brought about by maintaining balance through several steps.

First, dividing the deployment process into distinct stages helps mitigate the risk of a single point of failure. If deployment is done in incrementally smaller manageable phases, whatever issues arise can only be scoped within the bounds or perimeter of certain phases, and diagnosing where the problem is for a fix becomes so much easier and faster. As shown in Figure 34, It helps streamline the process, and tasks can run iteratively. For example, with differentiation in setup, security hardening, functional testing, and final deployment into different stages, each stage can be executed independently and concurrently, thus optimizing the timeline for deployment.


corrected pre audit

running skander-lahbaiel created pipeline for commit 074da78b created 8 hours ago

For dev-test

latest 6 Jobs In progress, queued for 2 seconds

Figure 34: CI/CD stages.

Proper resource usage management is key. Once deployed, the resources consumed during the security and performance tests need to be cleaned, that is, data erased. Normally, this would include the termination of all temporary instance usage, memory, or CPU resources allocated and, in general, the deletion of any data or artifacts used during the tests. The automation scripts for cleanup reduce the resource footprint of the security hardening flow in a way that does not affect the normal operations of the cluster. This approach retains resource availability and maintains the runtime efficiency of deployed applications.

To conclude, CIS hardening can be effectively integrated into containerized application deployment without impacting deployment time or resource usage by dividing the process into distinct stages and managing resources efficiently. This approach ensures enhanced security while maintaining performance and efficiency.

6. DISCUSSION

6.1. Implications for Research

The conducted study has several implications for research. By providing empirical evidence of the benefits of automated security testing and CIS hardening in a real-world scenario, the study contributes to the body of knowledge on cloud-native security practices. It validates the importance of automated security checks and continuous compliance, demonstrating their effectiveness in maintaining a secure and resilient application environment. This study serves as a framework that other researchers can follow, offering detailed documentation and practical insights into implementing similar security enhancements.

The study also shows that not all security remediations and hardening steps can be easily fully automated, and some should be done manually. This opens the door for future work to investigate if a way exists to be able to fully automate all security and hardening remediations.

6.2. Implications for Industry

6.2.1. Improved Security Posture

Integrating automated security testing into the CI/CD pipeline significantly enhances the security posture of NextCloud. Regular execution of CIS benchmarks ensures continuous monitoring and enforcement of security standards, leading to early identification and remediation of vulnerabilities. This proactive approach aligns with industry best practices and reduces the risk of security breaches.

6.2.2. Early Detection of Issues

Automated testing facilitates the early detection of issues during the development cycle, minimizing the impact of deployment misconfigurations and reducing the cost and effort required for remediation. Early detection of security vulnerabilities ensures that these issues are resolved before they affect end users, maintaining application stability and reliability.

6.2.3. Increased Confidence in Core Functionalities

Implementing automated functional tests increases confidence in the reliability of NextCloud's core functionalities. These tests simulate user interactions and validate that critical features work as intended, reducing the risk of introducing defects into the production environment.

6.2.4. Enhanced Compliance

Continuous integration of compliance checks within the CI/CD pipeline ensures adherence to industry standards and regulatory requirements. This proactive compliance monitoring simplifies audits and reduces the risk of regulatory penalties, enhancing the organization's reputation and building trust with users and stakeholders.

6.3. Threats to Validity

6.3.1. Internal Validity Threats

Continuous interaction with specific stakeholders could introduce bias or limit the exploration of alternative approaches. To mitigate this, the study engaged a diverse range of stakeholders, including a networking expert, a system administrator, and cloud expert, ensuring a well-rounded perspective.

6.3.2. External Validity Threats

The generalizability of the findings may be limited to similar cloud-native environments. However, the use of standardized tools and benchmarks, such as kube-bench, Ubuntu Security Guide and Sonobuoy may enhance the applicability of the results to other contexts.

6.3.3. Construct Validity Threats

The use of technical jargon and complex terminology could pose a threat to construct validity. This was addressed by providing comprehensive documentation and clear explanations of the concepts and procedures used in the study, ensuring that the findings are accessible to a broad audience.

6.3.4. Reliability Validity Threats

The reliability of the study was ensured through multiple runs of the tests and thorough review of configurations and scripts. Regular audits and validation checks were conducted to confirm the consistency of the results, enhancing the reliability of the findings.

In summary the integration of CIS hardening and automated security and functional testing into the CI/CD pipeline of NextCloud has proven effective in enhancing security and compliance without impacting other metrics. The study provides insights for both researchers and industry practitioners, highlighting the importance of automation in maintaining secure and functional cloud-native applications. The approach of dividing deployment into stages, automating resource cleanup, and ensuring consistent application of security configurations proved that security measures can be implemented without compromising deployment time and resource usage. This study sets a precedent for future research and industry practices, emphasizing the critical role of automated security and functional tests in the continuous delivery of secure and resilient cloud-native applications.

7. CONCLUSIONS

This study investigated the integration of the functional, conformance test as well as the security auditing and hardening in the CI/CD pipeline of the deployment of NextCloud on bare metal cloud infrastructure. The project also explored the use of automated documentation for maintaining consistency, clarity and remediating the complexities of the project. Using Design Science Research methodology, the project was divided into three main iterations, each addressing specific aspects of deployment, testing, and security hardening. Throughout the project, significant improvements in quality and security metrics were observed following the integration of automated testing into NextCloud's CI/CD pipeline. Initial audits based on CIS benchmarks revealed a high number of vulnerabilities and non-compliance issues. However, after executing security-hardening measures, the compliance scores improved dramatically, showcasing the effectiveness of the automated testing and remediation steps built into the CI/CD pipeline. These findings highlight the critical role of automation in maintaining the security and reliability of cloud-native applications.

The study also identified several challenges associated with integrating automated tests into the CI/CD pipeline, such as the complexity of setup and configuration, increased pipeline execution time, and issues related to UI interactions in functional tests. Mitigating these challenges involved detailed documentation, employing configuration management tools, and refactoring workflows to include testing stages. This approach ensured seamless integration of automated testing without disrupting existing processes. The primary benefits of integrating automated security and functional testing into the CI/CD pipeline included an improved security posture, early detection of issues, increased confidence in core functionalities, and enhanced compliance. Regularly running CIS benchmarks ensured continuous monitoring and enforcement of security standards, while early detection of functional failures and security vulnerabilities minimized the impact of failures. Automated functional tests validated critical features, reducing the risk of introducing defects into the production environment.

CIS hardening was effectively integrated into the deployment process of NextCloud without negatively impacting key performance metrics such as deployment time and resource usage. This was achieved through modular deployment stages, automated resource management, and consistent application of security configurations. The strategic approach ensured that robust security measures were implemented efficiently, maintaining a balance between security and performance. Future work for LiJo Technologies could focus on integrating advanced security measures such as AI-driven threat detection and response systems, improving scalability solutions to handle larger datasets and increased user load, enhancing user experience through more intuitive interfaces and additional features, expanding the CI/CD pipeline to include more test scenarios.

For the industry, adopting CIS hardening at the build and release life cycle of containerized applications is expected to guarantee a better security posture without trade-offs relating to deployment time, resource usage, and runtime efficiency. This balance can be maintained through several steps. First, dividing the deployment process into distinct stages helps mitigate the risk of a single point of failure. By conducting deployment in incrementally smaller, manageable phases, issues can be isolated within specific stages, making diagnosis and fixes easier and faster. This approach streamlines the process and allows reducing the effects of failing stages, optimizing the timeline for deployment. Proper resource usage management is also crucial. Once deployed, resources consumed during security, conformance and functional tests need to be cleaned up. This includes terminating temporary instances, releasing memory or CPU resources, and deleting any data or artifacts used during tests. Automation scripts for cleanup reduce the resource footprint of the security hardening flow, ensuring that normal cluster operations are not affected. This retains resource availability and maintains runtime efficiency. Also, integrating CIS hardening within the CI/CD pipeline automates tests and configurations, ensuring consistency across different environments. Automation reduces the chances of human error, allowing continuous compliance with security benchmarks and maintaining optimal performance metrics. Thus, to explain the strategy to integrate CIS hardening properly within the development and deployment processes we should involve a mix of modular deployment stages, automatically managed resources, and consistent application of security configurations. This approach ensures that security enhancements are implemented without compromising deployment time, resource usage, or runtime efficiency, maintaining both balance and security in the operational environment. Same as for the functional tests, we should ensure the tests simulating user interaction are being performed on the same exact environment. One way is the strategy implemented in this paper, to automatically create containers in which the testing script is executed.

In conclusion, the integration of CIS hardening and automated functional and conformance testing into the CI/CD pipeline significantly enhanced the security, and compliance of the NextCloud deployment and made the team more confident about the functionality and conformity of the system. This study provides valuable insights for both researchers and industry practitioners, emphasizing the importance of automation in maintaining secure and efficient cloud-native applications. The findings underscore the necessity of continuous security monitoring and the benefits of early issue detection in delivering secure and resilient applications. Future work in this domain should focus on further enhancing security measures by addressing threats that cannot be automated such as social engineering and insider threats. Also automating remediations that were not automated by this study. Additionally, expanding the scope of the audits to include compliance with other standards such as NIST and ISO/IEC 27001 will further strengthen the security posture of the system. For the private cloud hosting the application, disaster recovery strategies could be implemented such having a larger geographic distribution and performing regular backups.

REFERENCES

[1] N. Kratzke and P.-C. Quint, "Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study," *Journal of Systems and Software,* p. 16, January 2017.

[2] M. Shahin, M. A. Babar, M. Zahedi and L. Zhu, "Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges," in *11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* , Toronto, Canada, 2017.

[3] D. Bharadwaj and B. S. Premananda, "Transition of Cloud Computing from Traditional Applications to the Cloud Native Approach," in *North Karnataka Subsection Flagship International Conference (NKCon), IEEE*, 2022.

[4] T. Rose and X. Zhou, "System Hardening for Infrastructure as a Service (IaaS)," in *2020 IEEE Systems Security Symposium (SSS)*, 2020.

[5] T. Rangnau, R. v. Buijten, F. Fransen and F. Turkmen, "Leveraging cloud capabilities is a key component to optimize modern software solutions, emphasizing continuous integration and continuous deployment (CI/CD), removing silos between development and operations are key to ensure that a project meets competit," in *IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, 2020.

[6] "NextCloud," 10 05 2024. [Online]. Available: https://en.wikipedia.org/wiki/Nextcloud.

[7] "What is cloud computing?," [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing/.

[8] "Google Cloud," 2024. [Online]. Available: https://cloud.google.com/learn/what-is-cloud-computing.

[9] "Proxmox," 25 April 2024. [Online]. Available: https://pve.proxmox.com/pve-docs/. [Accessed 24 May 2024].

[10] "Canonical Juju," Canonical, August 2023. [Online]. Available: https://juju.is/docs/juju. [Accessed 2024].

[11] "Kubernetes," Kubernetes, 19 September 2023. [Online]. Available: https://kubernetes.io/docs/concepts/overview/. [Accessed 24 May 2024].

[12] "Microk8s," Canonical, January 2024. [Online]. Available: https://microk8s.io/docs. [Accessed 24 May 2024].

[13] "LXC Containers," Canonical, 23 May 2024. [Online]. Available: https://ubuntu.com/server/docs/lxc-containers. [Accessed 24 May 2024].

[14] M. Huttermann, DevOps for Developers, Apress, 2012.

[15] M. Gokarna and R. Singh, "DevOps: A Historical Review and Future Works," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2021.

[16] C. Cowell, N. Lotz and C. Timberlake, Automating DevOps with GitLab CI/CD Pipelines, Packt Publishing, 2023.

[17] G. A. D. Lucca and A. R. Fasolino, "Web Application Testing," *ResearchGate,* p. 43, March 2006.

[18] L. J. Carnahan, "Conformance Testing," NIST, 25 August 2016. [Online]. Available: https://www.nist.gov/itl/ssd/information-systems-group/conformance-testing.

[19] IBM, "What are CIS benchmarks ?," [Online]. Available: https://www.ibm.com/topics/cis-benchmarks.

[20] Sonobuoy, "Documentation," [Online]. Available: https://sonobuoy.io/docs/v0.51.0/.

[21] J. Angara and S. Gutta, "The Factors Driving Testing in DevOps Setting- A systematic Literature Survey," *Indian Journal of Science and Technology,* p. 48, 2017.

[22] S. Vilkomir, W. Jenkins, P. Sharma and G. Pirocanac, "Framework for testing cloud platforms and infrastructure," in *International Conference on Cloud and Service Computing*, 2011.

[23] T. Rangnau, R. v. Buijtenen, F. Fransen and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," in *2020 IEEE 24th International Enterprise Distributed Object Computing COnference (EDOC)*, 2020.

[24] A. M. Putra and H. Kabetta, "Implementation of DevSecOps bu integrating Static and Dynamic Security Pipelines," in *2020 IEEE International Conference of Computer Science and Information Technology*, Laguboti, North Sumatra, Indonesia, 2022.

[25] J. v. Brocke, A. Hevner and A. Maedche, "Introduction to Design Science Research," in *Design Science Research Cases*, 2020, p. 13.

[26] NextCloud, "Using NextCloud as an on-premise solution for office and file sync share," Sorbonne University, Paris, 2019.

[27] J. Schmitt, "circleci," circleci, 2023 March 2023. [Online]. Available: https://circleci.com/blog/what-is-helm/. [Accessed 2 May 2024].

[28] Canonical, "Github," Canonical, [Online]. Available: https://github.com/canonical/lxd. [Accessed 04 May 2024].