Implementing Nextcloud-Based Solutions for Personal Cloud Storage: A Case Study

By
Muhammad Mubashar Shahzad
[SM3600012]

Report submitted in partial fulfillment of
the requirements for the project of

Cloud Computing

DEPARTMENT OF MATHEMATICS AND GEOSCIENCES

UNIVERSITY OF TRIESTE

2024

TABLE OF CONTENTS

ABSTRACT

Today, the world witnesses a constant surge in the creation and use of information across different realms, from personal lives to business operations, owing to globalization. This necessitates the perpetual availability of information, crucial for business continuity and extending to individuals' private domains due to the distributed nature of information storage today. However, ensuring ubiquitous data access, regardless of location or device, presents a significant challenge.

Many businesses lack the resources and expertise to establish such infrastructure independently, leading them to adopt cloud-based storage solutions offered by service providers. While services from industry giants like Dropbox, Google, and Amazon are popular, concerns persist regarding data security and sovereignty. Precisely determining the physical location of data, despite claims of GDPR compliance, remains difficult, and transparency regarding cybersecurity measures is often lacking, particularly for entities handling sensitive information.

This underscores the need for innovative cloud storage solutions that empower users with greater control over their data and infrastructure. The aim of this report is to explore the design and implementation of such solutions, focusing on aspects like data management and protection.

Open-source platforms offer a promising avenue for this endeavor, given their cost-effectiveness, adaptability, and collaborative nature. Nextcloud, a leading open-source cloud storage solution, serves as the focal point of this investigation.

The aim of this report is to outline the development, execution, and rollout of a cloud-based file storage system. It incorporates integrated functionalities for user authentication and authorization management, file operation management, and bolstering security measures within the system. Nextcloud facilitates users in securely performing various file operations like uploading, downloading, deleting, and sharing files from their personal storage space. Encryption options are provided to ensure data protection during both storage and transmission processes.

Chapter 1

INTRODUCTION

In his book ”The Road Ahead” (1995), Bill Gates envisioned a future where individuals could conduct business, pursue education, explore global cultures, attend events, forge new connections, shop, and share memoriesall from the comfort of their own homes. This prediction has long since become reality.

Today, it’s rare to find someone who doesn’t rely on modern technology, particularly smart phones, computers, and the internet. Information Technology (IT) is so ingrained in our lives that ”Googling” has become synonymous with searching for information. IT has become indispensable in modern society, offering both advantages and drawbacks.

On the surface, IT simplifies many aspects of daily life. However, its underlying complexity can be daunting for many. Compared to the past, we now enjoy countless opportunities: remote work and study, global online shopping with doorstep delivery, and effortless cloud-based storage for our digital memories.

The COVID-19 pandemic further accelerated this trend, compelling millions to adapt to remote work for extended periods.

1.1 Nextcloud

This report aims to outline the design, execution, and roll-out of a cloud-based file storage system. Nextcloud emerges as a prominent choice, being an open-source solution renowned for its capabilities in file storage, synchronization, and sharing, among other functionalities. Leveraging Nextcloud’s robust features, the system was developed and implemented.

Nextcloud’s user authentication and authorization management offer substantial capabilities. It facilitates secure user registration, login, and logout processes. Moreover, it supports distinct user roles, such as regular users and administrators, aligning well with the need for role differentiation. Regular users benefit from individual storage spaces, ensuring data segregation and privacy, while administrators wield comprehensive management tools, facilitating effective user oversight. Categorizing users based on their usage patterns

could potentially optimize system performance.

In terms of file operations management, Nextcloud empowers users to conduct various tasks like uploading, downloading, deleting, and sharing files within their private storage areas. These operations are executed securely, with encryption options available to safeguard data during storage and transmission. Nextcloud's emphasis on security is evident through features like end-to-end encryption and robust user authentication mechanisms. It provides administrators with access control tools to define precise permissions, mitigating the risk of unauthorized data access. Server-side data encryption at rest fortifies data protection against unauthorized intrusions, while client-side authentication measures like multi-factor authentication and password complexity requirements further bolster system security.

Moreover, the report discusses the potential for augmenting security through the integration of third-party tools and explores various deployment options. This comprehensive approach ensures a robust and secure cloud-based file storage system.

1.2 Nextcloud User Access Control and Authentication

Nextcloud provides readily available built-in features for user authentication and authorization, allowing for flexible signup and login policies. These features include:

1.2.1 Registration and Login

Nextcloud offers a straightforward registration and login process, enabling users to seamlessly sign up, log in, or log out. Additionally, in the absence of an admin user, the first user to access the service becomes the admin. While this might initially raise concerns, providers typically configure the service before public availability to address this issue.

1.2.2 User Roles and Access Control

Nextcloud inherently supports different user roles, such as regular users and administrators. This policy ensures that each user is assigned appropriate privileges, maintaining the integrity of private user spaces.

1.2.3 Private Storage

By default, Nextcloud allocates a private storage space to each user, ensuring isolation between users. Furthermore, each user is assigned a predefined storage space, typically set to 512MB by default but easily adjustable through configuration settings.

1.2.4 Administrative Functions

Nextcloud provides administrators with a user-friendly dashboard for managing the entire file system. Administrators can perform tasks such as enabling encryption, implementing security policies, managing user accounts (addition, limitation, deletion), and more.

1.2.5 Authentication Process

Upon authentication, Nextcloud issues an access token for future HTTP accesses, stored only on the client's system. Additionally, user passwords are encrypted and stored securely in the Nextcloud database.

1.2.6 Client Registration

By default, Nextcloud does not allow autonomous client registration; only administrators can create accounts. To enable client registration, administrators can install and activate the REGISTRATION app, accessible through the admin dashboard's User menu -¿ Applications section. This feature allows new clients to register using standard email-password credentials.

1.3 Security Measures

1.3.1 Ensuring Secure File Storage and Transmission

Nextcloud offers predefined secure settings to enhance system security. For sensitive user data, Nextcloud provides the option to enable server-side file encryption. While this may impact system performance, it prevents unauthorized access to data. However, encrypted files can still be shared through the Nextcloud interface but not directly from the server. Enabling encryption increases storage space requirements for each file. Administrators

can easily enable encryption through the web interface by accessing the Default Encryption module app under Administration Settings -¿ Administration Security -¿ Server-side encryption.

1.3.2 Enhancing User Authentication Security

Implementing robust authentication measures is crucial, including enforcing strong password policies such as incorporating numbers and symbols and periodically requiring password changes. Additionally, implementing two-factor authentication further enhances security. Regularly conducting Nextcloud Security Scans using tools like https://scan.nextcloud.com helps identify and address potential vulnerabilities. Monitoring suspicious or unauthorized access patterns through the Activity and Logging sections of the admin page is also recommended.

1.3.3 Monitoring System Usage

Utilizing Nextcloud's Usage Survey report allows for regular monitoring of users, data storage, and activities. Managing system resources effectively is important due to the pay-as-you-go policy, where resource allocation directly impacts costs. Implementing maximum storage space limits for users and creating user groups (e.g., base-premium) with varying storage space allocations can help manage system resources efficiently.

1.3.4 Implementing Scheduled Updates and Backups

Regularly scheduling system updates and backups is critical for maintaining system security and data integrity. These updates ensure that the platform remains resilient against emerging threats and safeguards against data loss scenarios.

1.4 Cost-Efficiency Strategy

The selection of lightweight and scalable components such as Nextcloud, PostgreSQL, Redis, Prometheus, Node Exporter, AlertManager, Grafana, and Locust ensures an efficient and easily manageable system. Nextcloud's user-friendly features, including end-to-end

encryption and seamless collaboration capabilities, enhance data security without sacrificing usability.

Redis improves data access speed and reduces backend server loads, contributing to overall system efficiency. The monitoring tools - Prometheus, Node Exporter, AlertManager, and Grafana - play a vital role in maintaining system health and performance, making them indispensable components.

Locust is utilized during the development phase for load testing, identifying performance bottlenecks and optimizing resource allocation to promote cost efficiency. However, its usage is limited to this phase.

To further optimize cost efficiency, focus is placed on Nextcloud and PostgreSQL instances. Conducting a thorough analysis of system usage and sizing beforehand allows for informed decisions and implementation of appropriate measures. Fine-tuning monitoring tools based on data collection timing and user behavior enables efficient resource utilization.

1.5 Testing Approach

To facilitate convenient and well-documented testing, I chose to integrate the testing environment with Locust, a widely used Python package. Locust enables simulation of user interactions with the system at customizable workload rates, offering easy implementation and practical monitoring capabilities. Initially, I attempted to address this requirement using a local instance of Locust. However, I encountered numerous dependencies, highlighting the importance of utilizing a containerized instance for streamlined functionality. By containerizing Locust, I gained access to its instance via a web browser, allowing real-time monitoring of test results.

Chapter 2

DEPLOYMENT

The deployment was conducted within a laptop environment, specifically on an hp laptop equipped with 8 GB of RAM and a Core i5 CPU. The operating system utilized is Window 10. This setup provides a sufficient hardware foundation for executing various tasks and hosting the required software components.

2.1 Launching The System

The deployment infrastructure relies on Docker, utilizing Docker Compose as its core orchestration tool, simplifying the deployment process to a single command. By navigating to the main directory housing the docker-compose.yml file and executing ”docker-compose up -d,” the entire environment is brought to life seamlessly.
With this command, the following components are instantiated:

2.1.1 Nextcloud Container

Nextcloud is an open-source file sync and share software that allows users to store and access their files from anywhere. In the context of Docker, a Nextcloud container is an isolated instance of the Nextcloud application running within a Docker container.

Dedicated Volume for Data Persistence

In Docker, a volume is a specially designated directory within one or more containers that bypasses the Union File System. By configuring Nextcloud with its dedicated volume, data such as user files, configurations, and application data can persist even if the container is stopped or removed. This ensures that data remains intact and accessible across container restarts or updates.

2.1.2 MariaDB Container

MariaDB is a popular open-source relational database management system (RDBMS) that is compatible with MySQL. Within a Docker environment, a MariaDB container is an isolated instance of the MariaDB database server running within a Docker container.

Dedicated Volume for Data Integrity and Scalability

Similar to the Nextcloud container, the MariaDB container is configured with its dedicated volume. This volume stores MariaDB's database files, ensuring data integrity and consistency. Additionally, using a dedicated volume enables easy scaling of the database system by separating data storage from the container itself, allowing for efficient management of large datasets and high availability configurations.

2.1.3 Locust Container

Locust is an open-source load testing tool used to simulate user behavior and measure the performance of web applications. In Docker, a Locust container is an isolated instance of the Locust load testing tool running within a Docker container.

Utilized for Load Testing and Performance Evaluation

The Locust container is employed to generate simulated user traffic and stress test the Nextcloud and MariaDB containers. By mimicking user behavior, administrators can evaluate system performance, identify potential bottlenecks, and optimize resource allocation to ensure smooth operation under various loads.

2.2 Summary

To utilize the file storage system created for this project along with the files located in the root directory of the project, these steps can be followed:

- Launch Nextcloud:docker-compose.yml up -d docker-compose.yml ps
- Sign-in and install Nextcloud in the browser with the url localhost:8080
- Enable basic security measurements and nextcloud trusted domain.
- Log in with the default credentials: admin for user and password.
- Create users: ./create 30 users.sh
- Launch Locust: python -m locust -f locustfile.py
- Perform load testing with Locust: localhost:8089

Chapter 3
RESULTS AND DISCUSSION

To assess the system's capacity to handle increased loads, I employed Locust to replicate a spectrum of user interactions, providing invaluable insights into its performance under duress. This comprehensive evaluation unveiled the system's robustness in managing concurrent file downloads, demonstrating its resilience and suitability for real-world scenarios. Notably, the Locust testing encompassed an array of file sizes, illuminating the system's adaptability to diverse workloads and reinforcing its readiness for deployment in demanding environments.

3.1 Load Testing with Locust

3.1.1 Employing Locust for Load Testing

Locust served as our chosen tool to emulate user interactions, evaluating the scalability and efficacy of our Nextcloud deployment. User tasks were meticulously scripted in Python to mirror file download operations from the Nextcloud platform. As an open-source load testing utility, Locust facilitated the emulation of millions of simultaneous users, enabling rigorous assessment of our web application's performance and scalability. This evaluation was paramount in understanding how our system would perform under heavy loads, a critical factor in ensuring optimal functionality of our cloud-based file storage solution amidst anticipated user traffic surges.

3.2 Load Testing Results

The file system test was conducted locally on my personal computer, mimicking real-world usage conditions in a controlled environment.

The file system test was bifurcated into 4 segments: one focusing for 10 users second for 20 users, third for 30 users and forth is comarison of all these. Analysis of the results depicted in Figures 1, 2, 3 and 4 respectively.

However, transitioning to the large-sized file test revealed a notable decline in perfor-

mance. The system struggled to cope with the increased load, managing less than 2 requests per second and exhibiting an average response time of approximately 30 seconds. This stark contrast necessitates further investigation to identify the underlying cause, whether it be an architectural limitation or an implementation issue.

Despite the challenges encountered, the absence of server overload issues is encouraging, as indicated by the consistent absence of request failures. This instills confidence in attributing the performance bottleneck to architectural constraints.

Figure 3-1: Locust load testing results for a maximum of 10 users

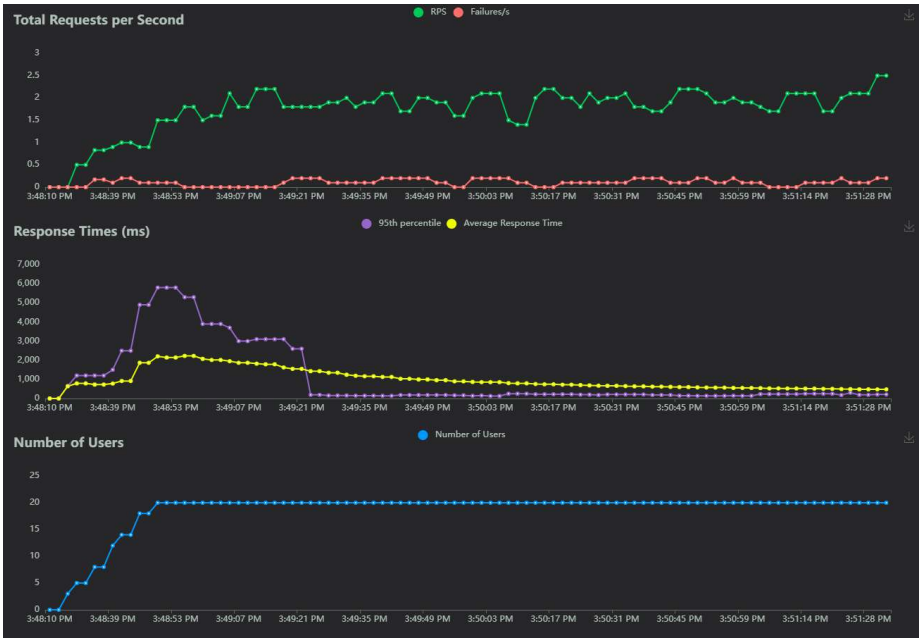

Figure 3-2: Locust Locust load testing results for a maximum of 20 users

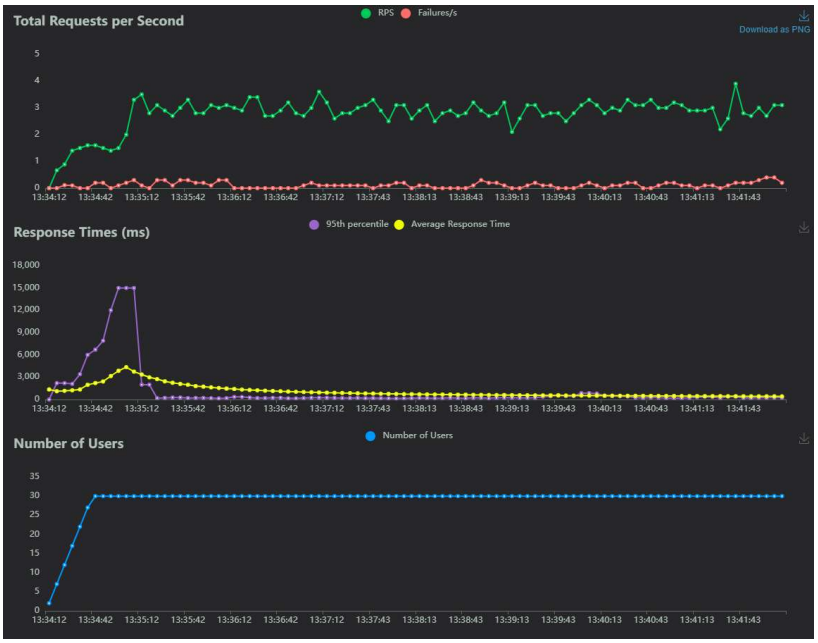

Figure 3-3: Locust load testing results for a maximum of 30 users

Figure 3-4: Comparison of 10 users, 20 users and 30 users performance

3.3 Cloud Deployment

Choosing a Cloud provider can inherently provide scalability and adaptability for your file system, presenting a robust solution for modern infrastructure requirements. One approach worth exploring involves leveraging a cloud database, harnessing the horizontal scaling capabilities inherent in cloud platforms to effectively distribute the infrastructure load. Data can be securely stored within an Object File System, such as Amazon S3, ensuring seamless scalability and ample storage capacities. To efficiently manage fluctuating workloads and optimize deployment costs, utilizing the autoscaling functionality of a cloud provider is essential.

The decision to use Amazon S3 as the storage system is rooted in the extensive capabilities offered by AWS services. Amazon Web Services boasts robust autoscaling features, facilitating dynamic resource adjustments based on infrastructure demand. Additionally, the composability principle of its Object File System enables effortless scaling of specific resources as required. Moreover, being founded on an Object Storage System, it excels in handling large file sizes, effortlessly providing backup and replication options. Furthermore, it offers versatile database systems that can serve as a backend for platforms like Nextcloud.

3.3.1 Advantages

Scalability

Cloud providers typically offer seamless scalability, enabling rapid adaptation to user demands and varying workloads.

Managed services

Utilizing a cloud provider can alleviate the burden of infrastructure and database management through comprehensive service offerings.

Global Accessibility

By leveraging a cloud provider, we tap into a network of worldwide data centers, ensuring global accessibility with consistent availability and reduced latency based on geographic

location.

Cost Efficiency

Cloud deployment operates on a pay-as-you-go model, offering cost efficiencies particularly beneficial for organizations with fluctuating workload demands.

3.4 Conclusion

In summary, this report detailed the implementation and evaluation of a Cloud File System utilizing Nextcloud, Docker, and Docker Compose. Nextcloud was selected for its robust feature set, intuitive interface, and scalability potential. The report delved into various aspects, covering user authentication and authorization, security protocols, database selections, storage alternatives, and scalability assessments.

Deployment was executed using Docker and Docker Compose, facilitating seamless container orchestration. Configuration adjustments were made to Nextcloud settings, including modifications to user storage allocations. Testing was performed using Locust, a Python library for simulating user interactions, to assess system performance across diverse workloads.

Findings indicated satisfactory performance in managing medium-sized files but identified challenges with handling large-sized files, suggesting potential architectural constraints.

Ultimately, scalability strategies were explored, encompassing deployment options on existing NFS/storage servers and cloud-based solutions such as Amazon S3. The determination between these alternatives hinges on factors such as current infrastructure, workload variability, and cost implications.

Chapter 4
Cloud-Based File Storage System in Kubernetes

This report details the process of setting up a Kubernetes cluster using Minikube on a Windows operating system, along with deploying applications using Helm.

4.1 Elements Explained

4.1.1 VirtualBox

VirtualBox is virtualization software that enables the creation and management of virtual machines on various operating systems. It is commonly used with Minikube to provide a virtualized environment for running Kubernetes clusters.

4.1.2 Kubernetes

Kubernetes is an open-source platform designed to automate deployment, scaling, and management of containerized applications. It facilitates efficient management of application lifecycles and ensures their availability and resilience.

4.1.3 Pods

Pods are the smallest deployable units in Kubernetes. They consist of one or more containers that share networking and storage resources. Pods are used to deploy and manage applications, with Kubernetes managing their creation, scaling, and termination.

4.1.4 Minikube

Minikube is a tool used to set up a local Kubernetes cluster. It allows developers to run a single-node Kubernetes cluster on their machines for development, testing, and learning purposes.

4.1.5 Helm Charts

Helm is a package manager for Kubernetes that simplifies application deployment and management. Helm charts are packages containing pre-configured Kubernetes resources,

such as deployments, services, and configurations, bundled together for easy installation onto a Kubernetes cluster.

4.1.6 Services

Kubernetes Services provide a consistent way to access and expose applications running within a cluster. They abstract the underlying network details and provide a stable endpoint for intercommunication between different parts of an application or between different applications.

4.2 Installation Steps

Automated Installation Script

Bash Script: Automated Installation

linkcolorRGB0, 102, 204

listings xcolor hyperref

Bash Script: Automated Installation

Install VirtualBox

[language=bash] sudo apt update sudo apt install -y virtualbox

Install Minikube

[language=bash] wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 chmod +x minikube-linux-amd64 sudo mv minikube-linux-amd64 /usr/local/bin/minikube

minikube start minikube status

Install kubectl

[language=bash] sudo apt-get update sudo apt-get install -y kubectl

Install Helm

[language=bash] wget https://get.helm.sh/helm-v3.7.1-linux-amd64.tar.gz tar -zxvf helm-v3.7.1-linux-amd64.tar.gz sudo mv linux-amd64/helm /usr/local/bin/

Verify Installations

[language=bash] minikube version kubectl version –client helm version

4.3 Deploying Applications

4.3.1 Installing Helm Charts

With Helm installed, applications can be deployed using Helm charts. For instance, a Helm chart for deploying an application can be installed onto the Kubernetes cluster.

4.3.2 Deploying Applications

Finally, applications are deployed onto the Kubernetes cluster by applying Kubernetes YAML files to create necessary resources.

4.4 Conclusion

In conclusion, we successfully set up a Kubernetes cluster using Minikube on our Windows machine. We installed the necessary tools, including Minikube, kubectl, and Helm, and deployed applications using Helm charts. This setup is ideal for local development and testing, enabling experimentation with Kubernetes features and application deployments in a controlled environment.

Chapter 5

Exercise 2: mpi service in Kubernetes

5.1 Introduction

This report documents the implementation of the OSU MPI Benchmarks on a Kubernetes cluster. The objective of this project was to measure the MPI latency using OSU benchmarks, specifically for point-to-point and collective operations, and to visualize the results. The process involved setting up a Kubernetes cluster, deploying the necessary resources, running the benchmarks, and interpreting the results.

5.2 Tools and Technologies Used

- **Docker**: Containerization of applications.
- **Kubernetes**: Orchestration of containerized applications.
- **Kubeflow MPI Operator**: Management of MPI workloads on Kubernetes.
- **OSU MPI Benchmarks**: Measurement of MPI performance.
- **Python and Matplotlib**: Data visualization.
- **Git**: Version control.
- **Minikube**: Local Kubernetes cluster.

5.3 Implementation

5.3.1 Setting Up the Environment

To set up the environment, the following commands were executed:

[language=bash] Install Minikube

curl -l minikube

https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 chmod +x

minikube sudo mv minikube /usr/local/bin/

Start Minikube

minikube start –driver=docker

Install kubectl

curl -l https:// dl.k8s.io/release/$(curl -L -s$

$https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectlchmod + xkubectl$

$sudomvkubectl/usr/local/bin/$

Verify installation

kubectl version –client

Two MPI benchmarks were implemented: point-to-point latency and broadcast latency. The YAML files used to create the Kubernetes resources are provided below.

5.3.2 Deployment YAML

[language=yaml, breaklines=true]osu-benchmarks-deployment.yaml

5.3.3 MPI Job YAML for Point-to-Point Latency

[language=yaml, breaklines=true]osu-benchmarks-mpijob.yaml breaklines=true]osu-benchmarks-deployment.yaml

5.3.4 MPI Job YAML for Broadcast Latency

[language=yaml, breaklines=true]mpi-bcast-job-diff-nodes.yaml breaklines=true]mpi-bcast-job-same-nodes.yaml

5.3.5 Building and Pushing Docker Image

The Docker image for OSU MPI Benchmarks was built and pushed to Docker Hub:

[language=bash] Build Docker image docker build -t mubasharshahzad/osu-benchmarks .

Push Docker image to Docker Hub docker push mubasharshahzad/osu-benchmarks

5.3.6 Deploying Resources on Kubernetes

The necessary Kubernetes resources were deployed using the following YAML files:

[language=bash] Create necessary directories mkdir -p /cloud-advance-project/k8s-manifests

Save the deployment YAML files nano /cloud-advance-project/k8s-manifests/osu-benchmarks-deployment.yaml nano /cloud-advance-project/k8s-manifests/osu-benchmarks-mpijob.yaml

Apply the YAML files kubectl apply -f /cloud-advance-project/k8s-manifests/osu-benchmarks-deployment.yaml kubectl apply -f /cloud-advance-project/k8s-manifests/osu-benchmarks-mpijob.yaml

5.3.7 Setting Up MPI Operator

The MPI Operator was set up as follows:

[language=bash] Clone the MPI Operator repository git clone https://github.com/kubeflow/mpi-operator.git

Checkout a stable version cd mpi-operator git checkout v0.3.0

Apply the MPI Operator YAML kubectl apply -f deploy/v2beta1/mpi-operator.yaml

5.4 Results and Graphical Representation

The OSU MPI Benchmarks provided latency measurements for point-to-point and collective operations. The results were visualized using Python and Matplotlib.

5.4.1 Results Interpretation

- **Point-to-Point Latency**: The latency for small message sizes (0-256 bytes) is relatively low, indicating efficient communication for small payloads. As the message size increases, latency increases logarithmically, highlighting the overhead associated with larger messages.

Broadcast Latency: The broadcast latency follows a similar trend, with low latency for small messages and increasing latency for larger messages. The overhead is slightly higher compared to point-to-point latency due to the nature of broadcast operations.

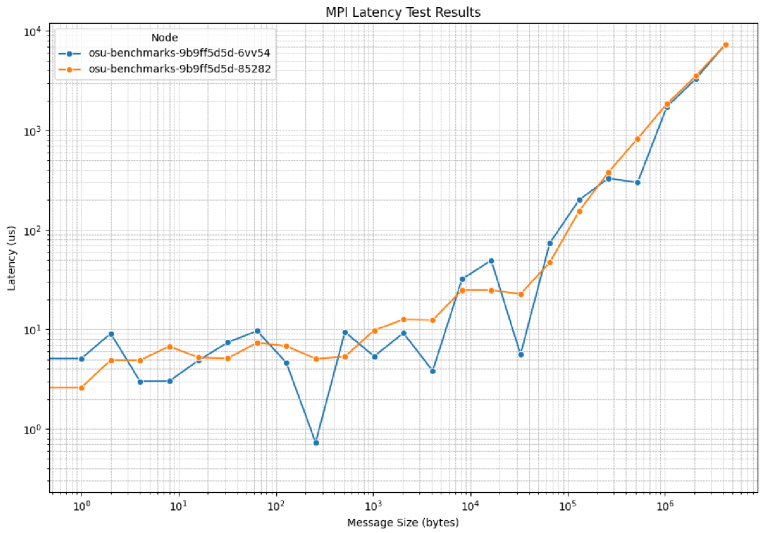

Figure 5-1: OSU MPI Point-to-Point Latency Test Results

5.4.2 Broadcast Latency

Below is the Python script used for plotting the broadcast latency results:

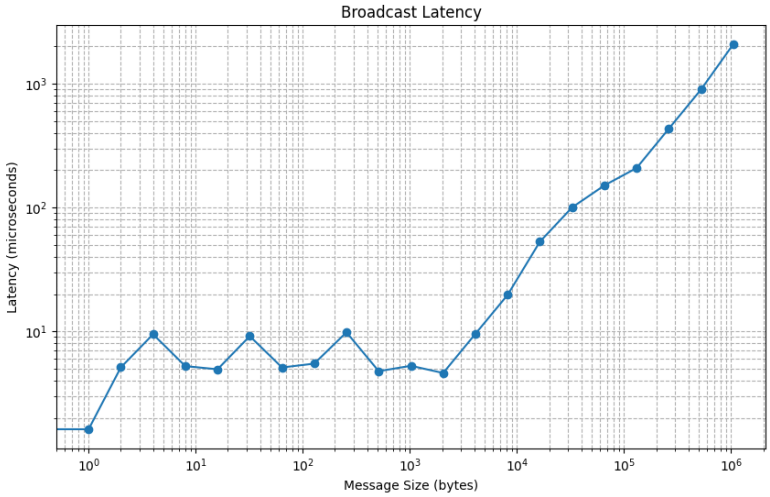

Figure 5-2: OSU MPI Broadcast Latency Test Results

5.5 Conclusion

In this project, we successfully deployed OSU MPI Benchmarks on a Kubernetes cluster using the Kubeflow MPI Operator. The benchmarks provided detailed latency measurements for different message sizes, both for point-to-point and broadcast operations.

5.5.1 Challenges and Difficulties

- **Resource Allocation**: Ensuring sufficient CPU and memory resources in the Minikube cluster to run the MPI jobs was challenging. Initial attempts failed due to insufficient CPU availability.
- **Custom Resource Definitions (CRD)**: Setting up and managing CRDs for the MPI jobs required careful configuration to avoid issues such as metadata size limitations.
- **Containerization and Deployment**: Building, pushing, and deploying Docker images required meticulous attention to configuration details to ensure compatibility and performance.

5.5.2 Key Takeaways

- Kubernetes, combined with Kubeflow, provides a robust platform for running MPI workloads, facilitating scalability and resource management.
- Proper resource allocation and cluster configuration are crucial for high-performance computing tasks.
- Visualizing performance metrics is essential for understanding and optimizing MPI implementations.