university of groningen / faculty of science and engineering

UNIVERSITY OF GRONINGEN

MASTER'S RESEARCH INTERNSHIP

Sandbox Code Runner

Author:
Mohammad al Shakoush

Supervisor(s):
Prof. Vasilios Andrikopoulos
Sahar Ahmadisakha, MSc

A Research Internship Report submitted in fulfillment of the requirements for the Master's Degree in Computing Science in the

Faculty of Science and Engineering
Department of Computing Science

September 25, 2024

Abstract

This document discusses the redesign and implementation of a new application for the "Cloud Computing and Cloud-based Applications" course at the University of Groningen (RUG). The course, essential to the Computer Science master's program, integrates both theoretical and practical components, focusing on cloud deployment and scalability.

The new application was designed to better align with the curriculum and the technologies students are familiar with, such as Java, Spring Boot, and Python. This approach simplifies the learning process by emphasizing cloud computing fundamentals and integrating tools that are widely used in the industry. Unlike the C# and .NET-based application used previously in the course, and which faced challenges due to technological incompatibility and poor documentation, this new application offers enhanced accessibility and a more streamlined experience, with intuitive interfaces and improved support resources for students.

A key enhancement in the new system is the introduction of "Virtual Labs" (VLs), cloud-based environments hosted on the RUG's OpenStack cluster. These labs fundamentally transform the learning experience by eliminating the need for local setup, providing students with immediate access to consistent, pre-configured environments. The automated deployment, powered by Terraform, not only ensures a smooth and standardized experience but also allows students to focus more on learning and less on technical setup, ultimately enhancing both efficiency and educational outcomes.

Contents

List of Figures

1 — Introduction

1.1 Background

This document provides an in-depth analysis of the design, development, and implementation of a new application developed for a critical component of the Cloud Computing and Cloud-based Applications course offered at the University of Groningen (RUG). As a master's level course, it plays a significant role within several academic tracks, particularly those focusing on cloud computing, distributed systems, and scalable application design. This course is also strategically positioned as a foundational prerequisite, equipping students with the necessary skills and knowledge for advanced courses in the curriculum.

The course content is divided into two main components: theoretical and practical. The theoretical component covers fundamental principles of cloud computing, including cloud service models, architecture, and security concerns. It also explores advanced topics such as cloud-native application development, microservices architecture, and continuous integration/continuous deployment (CI/CD) pipelines. This document specifically focuses on the practical component of the course, providing a detailed exploration of a target application that serves as the core project for this segment. This is designed to help students apply and implement the theoretical concepts they have learned.

The learning objectives associated with the practical component are comprehensive and aim to equip students with hands-on experience in transitioning an application from a local development environment to a fully operational cloud-based system. Specifically, students are tasked with deploying the application on the University's OpenStack cluster, a private cloud platform that provides a real-world environment for cloud computing exercises. A critical aspect of this deployment is ensuring that the application is capable of scaling to potentially serve millions of users, a requirement that underscores the importance of understanding cloud scalability and resource management.

The practical component, referred to in this document as *"the application"*, is purposefully designed with complexity to challenge students, but its modular structure ensures that it remains manageable and promotes effective learning. This modularity is crucial, as it allows for the seamless adoption of cloud technologies without necessitating modifications to the core source code. This design choice ensures that students can focus on cloud integration and deployment strategies rather than being encumbered by the need to refactor or re-engineer the application itself, at least for the first phases of the project.

The practical component is meticulously structured into several milestones, each representing a key phase in the overall project. The initial milestone involves *"dock-*

erizing" the application—encapsulating the software into Docker containers, which are then orchestrated using Docker Compose. This milestone introduces students to containerization, a fundamental skill in modern cloud computing, and emphasizes the importance of environment consistency and application portability.

Following the Docker Compose deployment, students advance to the next milestone, which involves converting the application's components into Helm charts. Helm, a package manager for Kubernetes, simplifies the deployment and management of applications within a Kubernetes cluster. This milestone not only familiarizes students with Kubernetes, an industry-standard platform for container orchestration, but also with Helm's templating and configuration management capabilities, which are essential for managing complex cloud-native applications.

The final milestone of the practical component requires students to use Terraform, an Infrastructure as Code (IaC) tool, to deploy the application on the University's OpenStack cluster. This milestone is particularly challenging as it involves establishing a highly available Kubernetes cluster across multiple virtual machines, complete with auto-scaling capabilities. This exercise is designed to simulate real-world cloud deployment scenarios, where students must manage the complexities of distributed systems, ensure fault tolerance, and optimize for performance and scalability.

This practical component, and the application at its core, represents an evolution of the course from its initial version as offered last academic year, where a different application was used. The new application is part of the University's broader initiative to enhance the Computer Science master's program at RUG, reflecting the institution's commitment to providing students with cutting-edge tools and experiences that are directly applicable to the rapidly evolving field of cloud computing.

Overall, this document not only outlines the technical aspects of the application and its role within the course but also situates it within the broader educational objectives of the Computer Science MSc programme. By providing students with a rigorous, hands-on project, the course aims to bridge the gap between theoretical knowledge and practical skills, preparing graduates to excel in both academic research and industry positions related to cloud computing and large-scale system design.

1.2 Problem Description

Problem 1 (P1) The previous application used by the course was created by a former teaching assistant (TA) to the course as a product for the *"Web and Cloud Computing"* course that served as the predecessor for the current course. However, the decision to develop the application using C# and the .NET framework introduced several challenges that ultimately affected the application's suitability for use in a course laboratory setting. Feedback from both students and TAs during the previous iteration of the course highlighted these issues, making it apparent that a complete redesign of the application was necessary to ensure it could function effectively within the course's framework. Which is the aim of this research internship.

More specifically, while the .NET framework is a robust and widely used platform, it is not extensively covered within the RUG's Computer Science curriculum, and C# is not currently part of the standard curriculum. This disconnect between the technologies used in the application and the tools and languages familiar to students seemingly presented a significant barrier to student engagement and learning. The project's reliance on .NET and C# meant that students had to invest considerable

time in learning these new technologies, detracting from their ability to focus on the core cloud computing concepts that the course aimed to teach.

Problem 2 (P2) Additionally, the project suffered from a lack of comprehensive documentation. The only guidance provided was a basic README file at the top level of the project, which offered little more than a cursory overview of the application. There were no design diagrams, no component descriptions, and no detailed explanations of the application's architecture or functionality. Moreover, critical aspects of working with .NET—such as locating environment variables, managing database connections, and handling connection strings—were not documented. This lack of documentation made it difficult for students to understand the application's inner workings, hindering their ability to extend or build upon the project, and ultimately limiting their learning experience.

Problem 3 (P3) Another significant shortcoming of the original application was the absence of a clear rationale for deploying it in a Kubernetes environment. The application did not present any obvious bottlenecks that would necessitate the use of Kubernetes, nor were there any tangible benefits of cloud infrastructure that were readily apparent to students. As a result, the value of deploying the application in a cloud-based Kubernetes environment was not evident, leading students to perceive the exercise as unnecessary and disconnected from the course's objectives. This lack of clarity around the purpose and benefits of using Kubernetes in the context of the application further undermined the effectiveness of the practical component.

Problem 4 (P4) Another issue encountered last year was the difficulty students faced in setting up the necessary technology to perform the practicals. Establishing a local Docker and Kubernetes development environment proved to be particularly challenging. Students struggled with installing and configuring the required software, and the diverse range of operating systems and hardware configurations further complicated the setup process. This difficulty significantly impeded the progress of the practical component, as many students were unable to get the environment running, leading to frustration and delays.

1.3 Envisioned solution

To summarize, it is imperative to enhance the students' experience regarding the practical component of the course. This improvement plan focuses on the following key objectives:

1. **Redesign and Rewrite the Base Application:** The base application will be rewritten and redesigned in a programming language that is more familiar to Computing Science students at the University of Groningen. The technology stack will also be updated to align with those more commonly used among the student body. Which handles P1.
2. **Provide Comprehensive Documentation:** Clear and detailed documentation will be created, outlining the technology used, the processes involved, and the interactions between different components. This will assist students in understanding the project architecture and workflow. Which handles P2.

3. **Provide a bottleneck:** The application will have a clear bottleneck that requires scaling in the cloud to lead students to perceive the exercise as necessary. Which handles P3
4. **Ensure Standardized Development Environments:** Students will be provided with ready-to-use development environments and machines (servers) on the OpenStack Clusters. This will ensure a standardized setup across all students, eliminating discrepancies in local configurations and promoting a uniform learning experience. Which handles P4

These objectives will serve as the foundation for the enhancement of the practical component of the course and will be referenced throughout the improvement plan.

In response to these challenges, and as part of this research internship, the application will not only be completely redesigned but will also be accompanied by the development of new tools aimed at creating isolated environments for each group of students on the University's OpenStack cluster. These environments, referred to as *"Virtual Labs"*, will adhere to the principles of cloud computing by providing a ready-made, cloud-hosted environment for each student group. This approach eliminates the need for local development environments, thereby resolving the issues related to software installation and configuration on personal machines. These Virtual Labs will provide a consistent and uniform environment for all students, ensuring that everyone has access to the necessary resources and tools to complete the practical component successfully.

Finally, the students encountered substantial difficulties due to the absence of necessary installations on their local machines. Many students lacked the hardware and software capabilities to run the application concurrently with a Kubernetes cluster, which is resource-intensive. These constraints prevented students from fully engaging with the practical exercises, as they were unable to simulate real-world cloud environments on their personal devices. This limitation underscored the need for a cloud-based solution that would remove these barriers and provide a more accessible, scalable environment for all students.

In conclusion, the challenges encountered with the original application—ranging from technological incompatibility to inadequate documentation and unclear objectives—highlighted the need for a comprehensive redesign. The new application and its accompanying tools, developed as part of this research internship, aim to address these issues by providing a cloud-native solution that aligns with the course's learning objectives and the technological landscape of the RUG.

2 — Methodology

2.1 Methods

To achieve these objectives, a new application will be developed using Java and the Spring Boot framework. Java is a more familiar language and is taught at the RUG in both the Object-Oriented Programming (OOP) and Advanced Object-Oriented Programming (AOOP) courses. This familiarity will enable students to understand the backend more effectively and make modifications to the code if necessary, allowing them to focus on critical aspects such as scalability and cloud integration rather than the obstacle of learning a new programming language.

To further enhance the educational value of the project, another part of the application will be written in Python, a language with which many students are also familiar. This dual-language approach will showcase the advantages of designing applications for a microservices architecture, where different components can be written in various languages or developed by separate teams. This method not only demonstrates the flexibility and modularity of microservices but also reduces confusion among students, of both components being deployed together, by clearly delineating the roles and responsibilities of each component. The process of containerizing these components using Docker and scaling them within a Kubernetes environment will be integral to the learning experience, offering students practical insights into modern cloud-based application development.

The project will be streamlined by removing certain features from the previous iteration, such as multitenancy and user authentication, as these do not contribute significantly to the course's learning objectives. Instead, the emphasis will be on core cloud computing concepts and the practical application of these concepts in a real-world scenario. To support this, a high-level diagram will be provided to give students and teaching assistants (TAs) a clear overview of the application's architecture and component interactions. Additionally, a second diagram will show how the application scales within a Kubernetes environment, with specific bottlenecks highlighted to help students understand the advantages of cloud adoption. These visual aids are intended to simplify complex ideas and enhance understanding.

2.2 Virtual Labs (VL)

To address the challenges associated with setting up local development environments, the course will implement *Virtual Labs (VL)*—isolated, cloud-based environments hosted on the University's OpenStack cluster. These Virtual Labs will provide each group of students with a pre-configured development environment that mirrors the production environment. By utilizing VLs, the need for students to in-

stall and configure complex software locally will be eliminated, thereby reducing the risk of technical issues and ensuring that all students can begin their practical work without delay.

The VLs will include all necessary tools and configurations, such as Docker, Kubernetes, and the Java and Python runtimes, pre-installed and ready to use. This approach will not only standardize the development environment across all students but will also align with the principles of cloud computing by leveraging cloud infrastructure to deliver scalable, on-demand resources. The VLs will also facilitate easier monitoring and support by TAs, as they will have access to a consistent environment when assisting students with troubleshooting and debugging.

This allows students to focus more on applying cloud computing concepts and less on overcoming technical hurdles, leading to a more effective and immersive learning experience. The use of VLs also prepares students for industry practices by familiarizing them with cloud-based workflows and infrastructure management, thus bridging the gap between academic learning and real-world applications.

In summary, the methodology adopted for this project emphasizes the use of familiar technologies, the simplification of the application architecture, and the deployment of cloud-based Virtual Labs to enhance the learning experience. By focusing on these areas, the course aims to provide students with a deeper understanding of cloud computing concepts and practical skills that are directly applicable to real-world scenarios.

3 — Results

3.1 Design

The designed application called ”Sandbox Code Runner” was meticulously engineered to ensure no coupling between the components. The application consists of three main components and three additional elements serving as data sources and a message queue. The components are decoupled in accordance with microservices architecture design guidelines, allowing each component to be scaled horizontally and independently. The main three components, depicted in Figure 8.1, are:

1. Frontend
2. Backend
3. Orchestrator

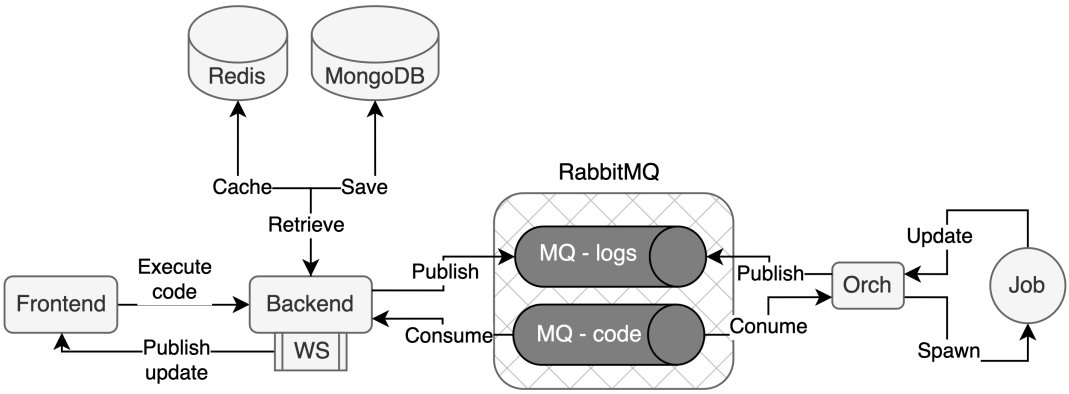

Figure 3.1: Sandbox Code Runner Architecture

3.1.1 Frontend

The frontend, developed using React JS, is responsible for handling user interactions and providing real-time feedback during code execution. It captures and processes user inputs, specifically Bash language code blocks, which are central to the system's operations. The frontend is structured around several key components such as `CodeBlockForm`, `LogsBox`, and `ExecutedBlocksList`, which collectively manage the user interface, code submission, and the display of execution results.

Before a code block is submitted, the frontend performs client-side validation to ensure the input is valid and that the backend service is available, preventing the submission of incomplete or erroneous requests. Upon successful validation, the code block is sent to the backend via a RESTful API, specifically the `/execute` endpoint.

The frontend then establishes a WebSocket connection using the `WebSocketClientContext` component, which receives real-time updates from the backend regarding the status and output of the code execution. These updates are dynamically rendered in the user interface to keep the user informed about the progress of their request.

The frontend is designed to be stateless, allowing multiple instances to be deployed behind a load balancer, which supports horizontal scaling and ensures that the system can handle a large number of concurrent users without performance degradation. This approach aligns with the overall architecture, as depicted in Figure 8.1, where the frontend communicates with the backend to publish updates and receive execution results efficiently.

3.1.2 Backend

The backend, implemented using Java and Spring Boot, serves as the core of the system's business logic and data handling. It provides several key functionalities through its RESTful API and WebSocket server. When a code block is received via the `/execute` endpoint, the backend validates the request using Data Transfer Objects (DTOs) such as `CodeBlockRequest` and `CodeBlockResponse`. This validation step ensures that the code block meets the required criteria before proceeding further.

Once validated, the backend assigns a unique ID to the code block and notifies the frontend with this ID, along with an initial WebSocket message to update the user interface. The validated code block is then forwarded to RabbitMQ using the `MessageProducer` class, where it enters the message queue system for execution.

The backend is also responsible for managing logs and metadata associated with each code block. This is handled by the `LogsManager` and `LogsMetadataManager` classes, which interface with both Redis and MongoDB for caching and persistent storage, respectively. The backend utilizes the Repository Pattern, implemented through interfaces such as `LogsRepository` and `LogsMetadataRepository`, to abstract data access and maintain clean, maintainable code. This pattern allows the backend to interact with the database layer in a consistent manner, as illustrated in Figure 8.1.

The backend's stateless architecture supports horizontal scaling, allowing it to handle increased load by deploying additional instances. This scalability is a critical aspect of the system's design, ensuring that the backend can process a large number of code execution requests efficiently.

3.1.3 Orchestrator

The orchestrator, implemented in Python, is responsible for managing the execution of code blocks within isolated Docker containers. It consumes messages from RabbitMQ, specifically from the `MQ - code` queue, as shown in Figure 8.1. The orchestrator retrieves the code block, spawns a new Docker container, and executes the code within this isolated environment.

The `orch.py` script is the core of the orchestrator, handling the lifecycle of each job from spawning the container to monitoring its execution. It interacts with the messaging system through a series of modules, including `message_handler.py` and `rabbit_config.py`, which configure the connection to RabbitMQ and manage message processing. Upon completion of the code execution, the orchestrator

updates the backend with the results, which include the exit status and any generated logs.

The results are then published to the `MQ - logs` queue, which are subsequently consumed by the backend and made available to the frontend. This flow ensures that the user receives real-time updates on the status and output of their submitted code, maintaining the responsiveness and reliability of the system.

The orchestrator's design is focused on scalability and fault tolerance. By utilizing Docker containers, the orchestrator can efficiently manage multiple concurrent jobs, with each job being isolated to prevent interference between code executions. Additionally, the use of RabbitMQ as a message broker ensures that messages are reliably delivered and processed in the correct order, contributing to the robustness of the system.

3.1.4 Design Decisions

The design of the Sandbox Code Runner was driven by the need to create a robust, scalable, and maintainable system capable of handling real-time code execution and logging in a cloud-native environment. Several key decisions were made to achieve these goals:

Message Handling in RabbitMQ

A significant design choice was the way messages are handled within RabbitMQ. Each message, representing a code block or log output, is acknowledged only after the entire execution process has been completed. This approach ensures reliable message processing and avoids the need for sticky sessions, which can complicate scalability and failover strategies. By treating each message as an atomic unit of work, the system maintains integrity even under high load, as each message is processed in its entirety before being acknowledged.

Data Caching and Persistence

The system employs both Redis and MongoDB to manage data efficiently. Logs, which represent the output of the containerized code execution, are stored in Redis for fast retrieval and in MongoDB for persistent storage. This dual storage strategy allows the system to leverage Redis's in-memory data store for quick access to frequently requested data, while MongoDB serves as the long-term data store. This design also simplifies fault tolerance, as Redis mirrors the data stored in MongoDB, ensuring that any data in MongoDB is readily available in Redis, thereby reducing latency in data access.

Fault Tolerance and Retry Mechanisms

The backend includes built-in retry mechanisms to handle transient failures, particularly in communication with RabbitMQ and the orchestrator. These retries are crucial for maintaining the system's robustness, ensuring that temporary network issues or service disruptions do not result in lost or unprocessed messages. The orchestrator, responsible for managing Docker containers that execute the code blocks, also communicates its status back to the backend. This communication includes updates on job progress and completion, which the backend then uses to update the

frontend in real-time. This ensures that the user is always informed of the current status of their code execution.

Separation of Concerns and Scalability

The architecture strictly adheres to the principles of microservices, with each component—frontend, backend, orchestrator, and message queue—being decoupled and independently scalable. This separation of concerns not only simplifies development and maintenance but also allows each component to scale independently based on demand. For instance, the frontend can scale horizontally to handle more users, while the backend and orchestrator can scale based on the number of code execution requests. RabbitMQ, as the central message broker, ensures that these components can communicate asynchronously, further enhancing the system's scalability and reliability.

Real-time Communication via WebSockets

WebSockets are used extensively to facilitate real-time communication between the backend and frontend. Unlike traditional HTTP, which follows a request-response model, WebSockets maintain a persistent connection, allowing continuous data flow. This is particularly beneficial for streaming log data from the backend to the frontend as soon as it is available. This real-time capability is essential for providing immediate feedback to users, enhancing the overall user experience.

These design decisions collectively ensure that the Sandbox Code Runner is robust, scalable, and capable of delivering a responsive user experience even under heavy load. The careful separation of concerns between different components, coupled with the use of modern technologies like RabbitMQ, Redis, MongoDB, and Docker, provides a solid foundation for the system's current functionality and future enhancements.

3.2 Technology Stack

The Sandbox Code Runner leverages a diverse and robust technology stack to ensure high performance, scalability, and ease of development. The key technologies used include:

- **Redis**: An in-memory data structure store used for caching log data, significantly reducing access times compared to querying the persistent database.
- **MongoDB**: A NoSQL database chosen for its scalability and flexibility, used to efficiently store and retrieve unstructured data.
- **Java and Spring Boot**: Java is a widely taught and robust programming language used in conjunction with the Spring Boot framework to develop the backend. Spring Boot simplifies the development of production-ready applications and provides a comprehensive suite of tools for building robust backend services.
- **Python**: Python is used for the orchestrator to demonstrate the flexibility and power of a microservices architecture, where different components can be implemented in various programming languages.

- **React JS**: React JS is employed for the frontend development. It is a popular JavaScript library for building user interfaces, known for its efficiency and flexibility in creating interactive and dynamic web applications.
- **WebSockets**: Used to facilitate real-time, bidirectional communication between the frontend and backend, maintaining a persistent connection for continuous data flow, which is crucial for real-time updates.
- **RabbitMQ**: A message broker that manages the communication between the backend and the orchestrator, allowing for distributed processing and task management. It is essential for handling the asynchronous nature of the system's tasks.
- **Docker**: Containerization technology used to package applications and their dependencies into containers, ensuring consistency across different environments and enabling scalable deployments.
- **Kubernetes**: Orchestrates the deployment, scaling, and operation of containerized applications, managing the backend, frontend, and orchestrator components to ensure the system can scale horizontally based on demand.
- **Ansible**: An automation tool used for provisioning and configuration management. Ansible ensures that the infrastructure is consistently deployed and configured across different environments, streamlining the process of scaling the system and maintaining its reliability.
- **Terraform**: An Infrastructure as Code (IaC) tool used to provision and manage cloud resources efficiently. Terraform enables the automated deployment of infrastructure components, ensuring consistency and scalability across different environments.
- **GitLab CI/CD**: GitLab's Continuous Integration/Continuous Deployment (CI/CD) pipelines are used to automate the building, testing, and deployment of the application. This integration ensures that new code changes are automatically tested and deployed, maintaining high code quality and enabling rapid iteration.

This technology stack provides a solid foundation for the system, enabling it to be scalable, resilient, and responsive to user demands, while also ensuring that the infrastructure and deployment processes are automated and efficient.

3.3 Scalability

The scalability of this system is achieved through a combination of architectural design choices, component decoupling, and the utilization of distributed technologies. Each component in the system—from the frontend to the backend, RabbitMQ, Redis, and MongoDB—is selected and configured to support horizontal scaling, ensuring that the system can handle increased load and expand its capacity as demand grows. This scalability is managed and facilitated by Kubernetes for orchestration and Ansible for provisioning and configuration management, which ensures consistent and automated deployments across different environments.

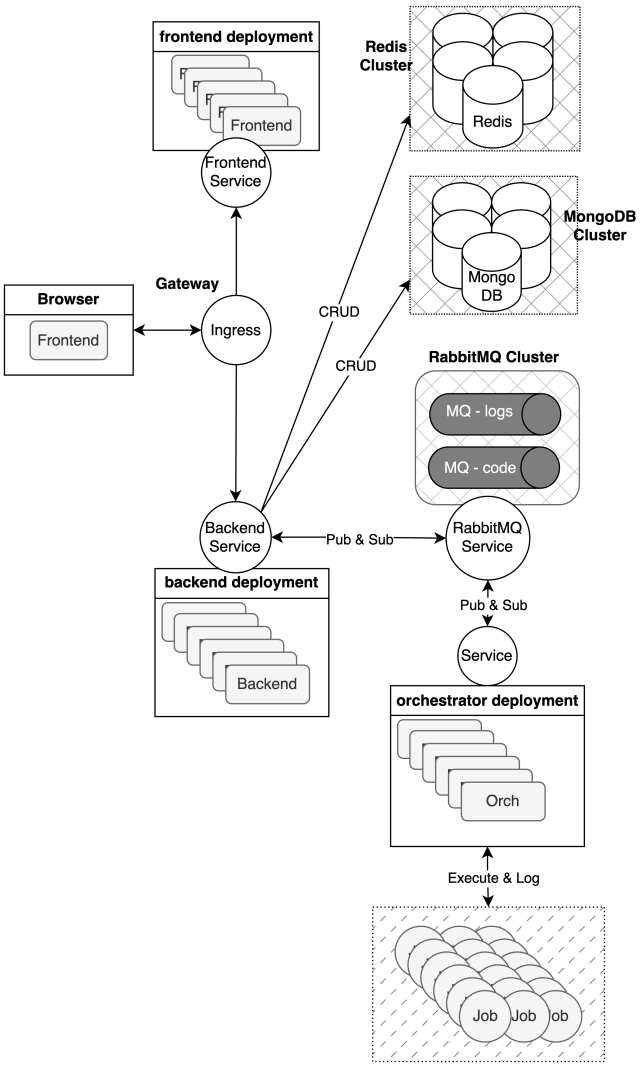

Figure 3.2: Scaling Diagram

3.3.1 Frontend Scalability

The frontend is inherently stateless, which allows for easy horizontal scaling. Multiple instances of the frontend can be deployed behind a load balancer, distributing incoming user requests across these instances. This design ensures that the system can support numerous simultaneous users without experiencing performance degradation. The use of WebSockets, which typically presents scaling challenges due to the persistent nature of connections, is managed through Kubernetes Horizontal Pod Autoscaler (HPA) configurations defined in the Helm charts. This configuration allows the frontend deployment to scale based on CPU or memory usage, dynamically adjusting the number of running instances to match the current load.

3.3.2 Backend Scalability

The backend is designed to be stateless and modular, which is critical for scaling. The stateless nature of the backend means that it can easily scale horizontally by adding more instances to handle increased requests. Docker is used to containerize the backend, and these containers are orchestrated using Kubernetes. The scaling

process is automatically managed by Kubernetes HPA. Ensuring that the backend deployment can handle fluctuations in demand without manual intervention.

The backend's reliance on RabbitMQ for handling code execution and log messages allows for distributed processing. RabbitMQ supports clustering, which means that the message queue itself can be scaled horizontally. This clustering is configured, allowing RabbitMQ to distribute messages across multiple nodes, which prevents any single node from becoming a bottleneck. Worker processes, in the form of backend instances, can then consume these messages concurrently, enhancing the system's ability to process a large volume of execution requests and log processing tasks simultaneously.

3.3.3 Message Queue Scalability

RabbitMQ is central to the scalability of this system. It acts as a broker for distributing tasks and log messages, enabling asynchronous processing and decoupling between the backend and the orchestrator. RabbitMQ's clustering capability is a key feature that supports its scalability, allowing multiple RabbitMQ nodes to work together, share the load, and provide fault tolerance. This configuration ensures that if one node fails, another can take over without disrupting the entire system.

Moreover, RabbitMQ supports exchange and queue federation, which enhances its scalability by allowing federated exchanges and queues to span multiple RabbitMQ clusters. This setup can be particularly useful in geographically distributed environments, ensuring low-latency message delivery and high availability even in large-scale systems.

3.3.4 Data Storage Scalability

Both Redis and MongoDB are chosen for their robust scalability features. Redis, acting as a cache layer, can be scaled horizontally by adding more nodes to a Redis cluster. This allows Redis to handle more read and write operations concurrently, which is crucial as the number of concurrent users and the volume of logs increase.

MongoDB, used for persistent storage, is a NoSQL database known for its scalability. MongoDB supports sharding, a process that distributes data across multiple servers, enabling the database to handle large datasets and high throughput. The sharding configuration, managed via Kubernetes, ensures that MongoDB can scale horizontally by adding more shards as the dataset grows. Additionally, MongoDB's replica sets distribute read operations across multiple nodes, reducing the load on any single node and contributing to overall system performance.

3.3.5 Orchestrator and Job Management Scalability

The orchestrator component, which manages the execution of code blocks in isolated Docker containers, is designed to scale with the number of incoming jobs. Kubernetes is used to dynamically allocate resources for job execution based on current demand, with HPA configurations automatically adjusting the number of orchestrator instances. This dynamic scaling capability is essential for maintaining performance and reliability, especially when handling numerous concurrent jobs.

3.3.6 System Scalability

In conclusion, the system's scalability is underpinned by its modular, decoupled architecture and the strategic use of technologies that support horizontal scaling and high availability. The use of Kubernetes for orchestration, combined with Ansible for provisioning, ensures that the system can dynamically adjust to changes in load and maintain performance under varying conditions. By distributing workloads across multiple instances, clusters, and nodes, the system can accommodate increasing user demands and data volumes without compromising performance or reliability. This scalability ensures that the system remains robust and responsive, even as it grows in size and complexity.

3.4 Infrastructure Automation and Deployment Workflow

The system's infrastructure and deployment processes are fully automated using a combination of Terraform, Ansible, and GitLab CI/CD. This integrated approach ensures that infrastructure is consistently provisioned, configured, and maintained across different environments, supporting the system's scalability, reliability, and operational efficiency.

3.4.1 Infrastructure as Code (IaC) with Terraform

Terraform is employed to define and provision the cloud infrastructure for the system using Infrastructure as Code (IaC) principles. This enables the infrastructure setup to be described in code, ensuring repeatable, consistent, and auditable deployments. Key components such as virtual machines, networking configurations, and security groups are defined within Terraform configuration files.

The state of the infrastructure, managed by Terraform, is securely stored and versioned in GitLab, allowing for collaborative infrastructure management and preventing conflicts. This setup is crucial for maintaining an accurate and reliable representation of the infrastructure across all environments.

3.4.2 Configuration Management with Ansible

Once the infrastructure is provisioned by Terraform, Ansible automates the configuration of servers and services. Ansible playbooks are executed to install necessary software, configure system settings, and deploy applications across the infrastructure. This automation ensures consistency across all servers, reducing the potential for configuration drift and minimizing manual intervention.

The playbooks are triggered as soon as the Master VM is provisioned, pulling the latest resources from GitLab, installing required packages such as Docker, and applying Helm charts to set up the environment. Ansible integrates seamlessly with GitLab CI/CD pipelines, ensuring that any configuration changes are automatically tested and deployed, maintaining a consistent and stable environment.

3.4.3 Continuous Integration and Deployment with GitLab CI/CD

GitLab CI/CD pipelines automate the deployment process, orchestrating the execution of Terraform and Ansible scripts whenever changes are made to the infrastructure or application code. These pipelines ensure that the system remains up to date with the latest configurations, applying changes in a controlled and reliable manner.

The CI/CD pipelines handle the entire lifecycle of the deployment, from building Docker images, pushing them to a container registry, to deploying the updated images on the OpenStack cluster. This approach ensures continuous integration and deployment, minimizing downtime and enhancing the reliability of the system.

3.4.4 Deployment Workflow and Environment Setup

The deployment workflow integrates multiple tools and repositories to effectively manage both the infrastructure and application components. The project is organized into several GitLab repositories, each corresponding to different system components such as the frontend, backend, orchestrator, and infrastructure resources. Each repository is equipped with its own CI/CD pipeline, automating the build, test, and deployment processes.

As illustrated in Figure 8.3, when a change is pushed to any of these repositories, the respective CI/CD pipeline is triggered. The pipeline builds Docker images, pushes them to a container registry, and deploys the updated images to the OpenStack cluster using Ansible scripts.

3.4.5 OpenStack Cluster Configuration

The application is hosted on an OpenStack cluster comprising a master VM and two worker VMs. Terraform scripts are used to provision this infrastructure, including the setup of networks, VM instances, and security groups. Once the VMs are up and running, Ansible playbooks configure the Kubernetes cluster on these VMs, which manages the deployment of the application containers.

The master VM orchestrates the Kubernetes cluster, while the worker VMs handle the application workloads. This configuration ensures that the system is scalable, resilient, and can be centrally managed through Kubernetes.

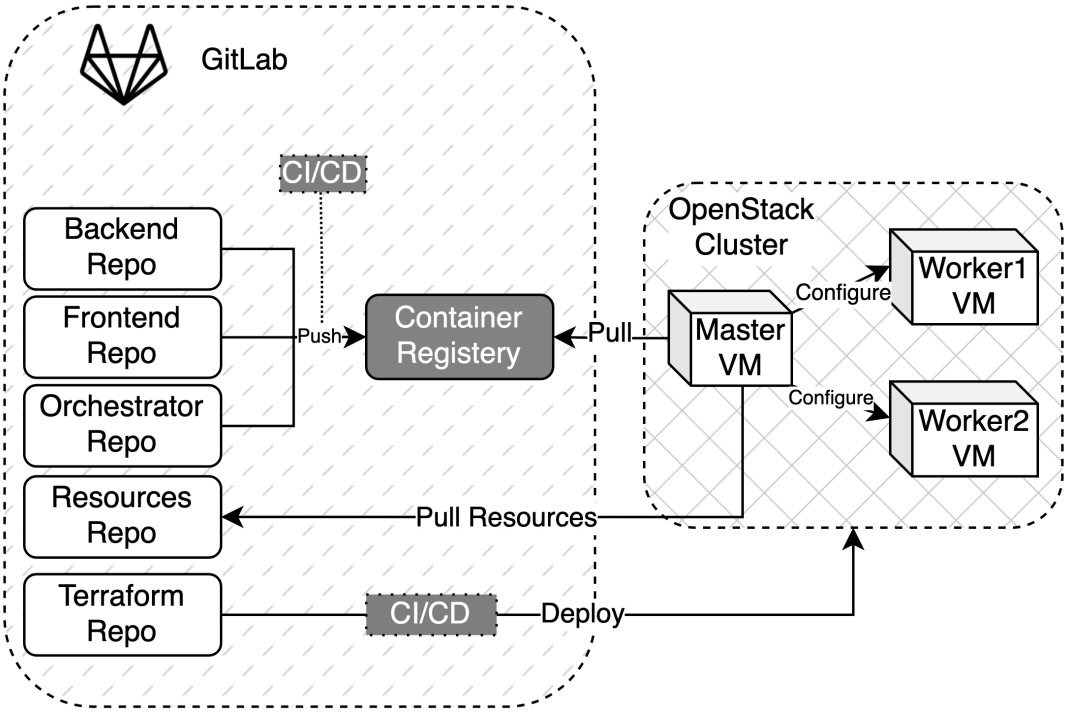

Figure 3.3: Deployed Application Overview

3.5 Virtual Labs

The Virtual Labs (VL) implementation is designed to streamline the deployment of isolated, cloud-based development environments for student groups, leveraging the University's OpenStack cluster. This approach ensures that each group of students has access to a consistent and fully configured environment, eliminating the need for local software installation and configuration, and aligning with the cloud computing principles taught in the course.

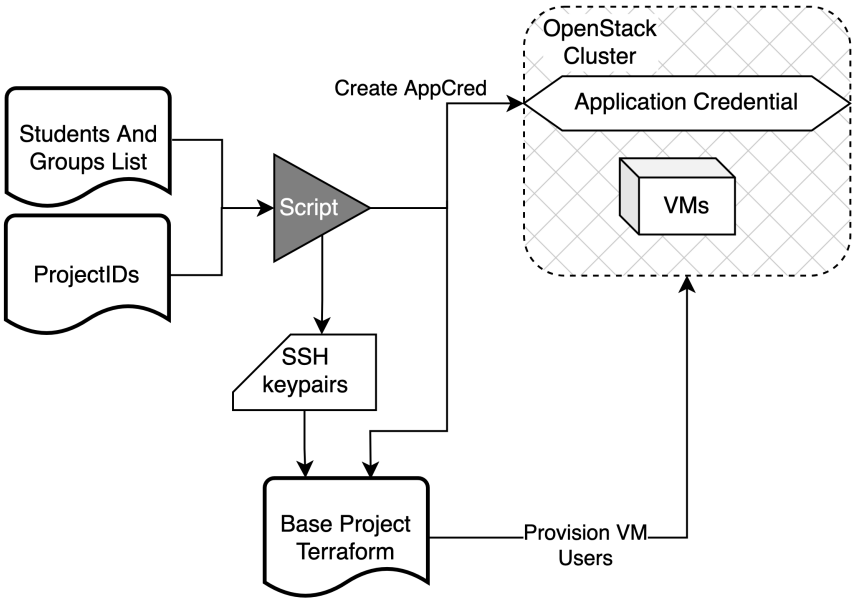

Figure 3.4: Virtual Labs

As depicted in Figure 8.4, the process begins with the provision of a CSV file that contains details about the students and their respective groups. This file is crucial as it serves as the basis for the subsequent steps in the Virtual Labs setup. The following steps outline the detailed process:

1. **Project Creation on OpenStack:** For each group listed in the CSV file, a dedicated project is created on the OpenStack cluster. This project acts as an isolated environment where the group can conduct their development activities. Each project is uniquely mapped to a group and assigned a corresponding project ID. This mapping ensures that the resources and environments are properly segregated and managed for each group, maintaining isolation and preventing cross-group interference.

2. **User SSH Key Pair Generation:** For each student, an SSH key pair is generated, with the public key being used to configure secure SSH access to the VM.

3. **Application Credentials (App Creds) Generation:** After the project creation, App Creds are generated for each project. These credentials are essential as they are the only possibility of interacting with the OpenStack API and resources. The generated App Creds allow for programmatic access to the project, enabling automated deployment and management of the environment via Terraform.

4. **Terraform-Based Deployment:** With the App Creds in place, the base Terraform project is deployed within each group's project on the OpenStack cluster. Terraform workspaces are utilized to manage the different group environments, allowing for scalable and repeatable infrastructure deployment. During this step, the necessary software is installed on the virtual machines (VMs) provisioned for each group. This software typically includes development tools such as Docker, Kubernetes, and the Java and Python runtimes, pre-configured to match the course requirements.

5. **User Account and SSH Key Pair Generation:** As the VMs are being created, individual user accounts are set up for each member of the group. This ensures that each student can securely access their group's environment using their personal credentials. The SSH key pair, along with the student's unique username (typically their student number, referred to as *snumber*), is critical for maintaining secure and personalized access to the cloud environment.

6. **Provisioning Access Details to Students:** Once the VM setup is complete, each student is provided with the necessary access details. This includes the floating IP address of their group's VM, their individual username, and their SSH key pair. These details are crucial for enabling students to connect to their Virtual Lab environment and begin their work. By providing these credentials directly, the process ensures that students can immediately start their practical exercises without the need for additional setup or troubleshooting, fostering a seamless transition to the cloud-based development environment.

This approach to Virtual Labs not only simplifies the technical setup for students but also enhances the scalability and manageability of the course's practical component. By automating the deployment process through Terraform and integrating secure access mechanisms, the course ensures that each student group has a robust and consistent environment tailored to their learning needs. The use of OpenStack's cloud

infrastructure further aligns the practical experience with real-world cloud computing practices, providing students with valuable hands-on experience in managing and deploying cloud-based applications.

4 — Discussion

4.1 Summary of Key Findings

The redesign and implementation of the new application for the "Cloud Computing and Cloud-based Applications" course yield several significant improvements that enhance both the educational and technical aspects of the course. These enhancements are carefully aligned with the course's objectives, preparing students more effectively for real-world cloud computing challenges.

One of the most impactful outcomes is the overall increase in student engagement and participation. This is achieved through several key innovations, including the introduction of Virtual Labs (VLs), the adoption of a modern and familiar technology stack, and the automation of infrastructure management. Together, these elements reduce the technical barriers that previously hindered students, allowing them to focus more fully on the core learning objectives.

Additionally, the uniform environment provided by the VLs enhances collaboration and teamwork among student groups. With all students operating within the same setup, troubleshooting and knowledge sharing become more straightforward, promoting a cohesive and productive learning environment. This consistency supports the development of collaborative skills that are essential in both academic and professional contexts.

The application leverages a technology stack centered around Java, Spring Boot, and Python—tools that are both widely taught in the academic setting and extensively used in the industry. This approach not only makes the application more accessible but also provides students with relevant, hands-on experience. The modular microservices architecture introduces them to essential concepts of cloud computing.

Infrastructure automation is another key area of focus. By integrating Infrastructure as Code (IaC) using Terraform and automating configurations with Ansible, the course introduces students to advanced DevOps practices critical for managing modern cloud infrastructures. These tools streamline the deployment process and provide students with practical experience in efficiently managing and scaling complex systems. The expectation is that students will make the application capable of potentially handling millions of users, reflecting real-world demands for scalable systems.

The architectural design of the application, emphasizing decoupled components and efficient message handling through RabbitMQ, results in a robust and reliable system. This design not only supports the educational objective of teaching scalability but also ensures that students understand how to build systems that are both fault-tolerant and capable of scaling horizontally, a crucial aspect of cloud-native applica-

tions.

In summary, the project successfully addresses the limitations of the previous system by introducing a range of technical and educational enhancements. These improvements not only meet the project's objectives but also significantly enrich the learning experience, equipping students with the skills and knowledge necessary to navigate and succeed in the complexities of cloud computing.

4.2 Challenges

The development of the new application was a significant undertaking, particularly as it was driven by a single student as part of a research internship. The process involved not just a change in the technology stack but a complete rethinking of the application's concept and structure.

One of the key challenges was the complete overhaul of the existing application. The previous system, based on .NET and C#, was not well-aligned with the course's educational goals or the students' technical backgrounds. The decision to shift to a new technology stack—centered around Java, Spring Boot, and Python—required a re-envisioning of the entire application. This was not just a technical upgrade but also a conceptual redesign, ensuring that the new application better served the course's focus on cloud computing and scalability.

Redesigning the application's architecture to meet these new goals involved careful planning. The new system needed to be modular and scalable, supporting the educational objective of teaching students about cloud scalability. This meant designing an application skeleton that students could extend to potentially handle millions of users, thereby giving them practical experience in building scalable solutions.

The introduction of VLs was another pivotal aspect of the redesign. These labs were intended to provide students with consistent, pre-configured cloud environments, which would eliminate many of the technical barriers encountered in previous iterations of the course. Developing these labs required addressing various technical issues related to cloud resource management, security, and network configurations. Ensuring that the VLs were both robust and easy for students to use was a significant part of the development process.

Managing the entire project as a solo developer required a high level of self-discipline and problem-solving ability. Balancing the need to redesign the application's concept, implement a new technology stack, and ensure the system's scalability and usability within the constraints of the project timeline was no small task. However, these challenges were met with a focus on delivering a functional, educationally effective tool that aligned with the course's objectives.

In summary, the development of the new application involved navigating several complex challenges, from conceptual redesign to technical implementation. The successful outcome of the project demonstrates the ability to effectively manage and overcome these challenges, resulting in a robust and innovative application that significantly enhances the learning experience.

4.3 Practical Implications

The redesigned application delivers substantial practical benefits, directly enhancing students' readiness for industry challenges. By incorporating industry-standard tools like Java, Spring Boot, Python, Kubernetes, and Terraform, the course now provides students with hands-on experience that closely mirrors real-world practices. This exposure not only bridges the gap between theoretical knowledge and application but also equips students with skills that are highly valued in the job market.

The introduction of VLs ensures that all students have equal access to a consistent, cloud-based environment, significantly reducing technical barriers and allowing them to focus on mastering cloud computing concepts. This uniformity fosters a more inclusive learning experience, making the course accessible to a wider range of students, regardless of their prior technical background.

A key focus on scalability within the course equips students with the ability to design and implement systems capable of handling large-scale operations, an essential skill in cloud computing. The requirement for students to extend the application to handle potentially millions of users instills a deep understanding of scalability, preparing them to tackle similar challenges in professional settings.

The integration of infrastructure automation tools like Terraform and Ansible introduces students to modern DevOps practices, providing them with practical experience in managing and automating cloud infrastructures. This knowledge is crucial for operating in environments where continuous integration and deployment are standard.

Moreover, the standardized setups facilitated by VLs promote effective teamwork and collaboration, essential skills in the software development industry. The course structure encourages students to work together within a consistent environment, enhancing both their technical and soft skills.

In summary, this project not only enhances the educational value of the course but also significantly boosts students' practical capabilities, better preparing them for careers in cloud computing and related fields.

4.4 Lessons Learned

Throughout the development of the new application, several key lessons emerged that are valuable for both future projects and educational design.

The decision to structure the application around a microservices architecture underlined the value of modularity. This approach not only made the application scalable, but also provided a clear, tangible way for students to interact with and learn about complex cloud systems. The modular design facilitated incremental learning, allowing students to build and extend the application in manageable steps, reinforcing their understanding of key cloud principles.

The introduction of VLs highlighted the critical impact of providing students with a consistent, ready-to-use environment. By eliminating setup challenges, the VLs allowed students to focus more on learning and experimentation, which led to better engagement and outcomes. This lesson reinforces the importance of simplifying

technical environments in educational settings to allow for a more focused and productive learning experience.

One lesson learned was the critical role of comprehensive and clear documentation in the success of the project. Detailed documentation not only supported the development process but also ensured that students could navigate the application and its components effectively. This experience highlighted that robust documentation is essential, especially in educational tools, as it helps bridge the gap between complex technology and the learner's understanding.

Finally, the project revealed the delicate balance required between introducing advanced technical concepts and maintaining accessibility for all students. Ensuring that the application was challenging enough to be instructive, yet approachable for those with varying levels of experience, was a key factor in its success. This balance is essential in educational design, where the goal is to push students' understanding without overwhelming them.

4.5 Recommendations for Future Research

While the new application has significantly enhanced the learning experience in the "Cloud Computing and Cloud-based Applications" course, there are several avenues for future research that could further improve its effectiveness and extend its impact.

Given the need to finalize the application before the course commenced, for example, there is significant potential for refining the application based on student feedback gathered during and after its initial use. Future research could focus on systematically collecting and analyzing this feedback to identify areas where the application can be improved. This might involve tweaking the user interface for better usability, adjusting the complexity of certain tasks to better match student skill levels, or enhancing the clarity of instructional materials. Incorporating real-world feedback from students who have used the application in a live educational setting will be crucial in making iterative improvements that enhance its effectiveness and accessibility.

In conclusion, the new application provides a solid foundation for cloud computing education, but there is potential for future research to expand its capabilities, enhance its impact, and adapt its principles to other areas of study.

5 — Conclusion

This paper detailed the redesign and implementation of a new application for the "Cloud Computing and Cloud-based Applications" course at the University of Groningen. The project successfully replaced the previous application with a modern, scalable one that aligns with current industry standards and educational goals.

Key innovations included the adoption of a microservices architecture and the introduction of Virtual Labs, which significantly enhanced the learning experience by providing a consistent, cloud-based environment for students. These improvements facilitate better engagement, practical learning, and collaboration among students.

The development process, managed by the author of this paper, involved overcoming several technical and conceptual challenges, ultimately resulting in a robust and effective educational tool. Lessons learned from this project emphasize the importance of aligning technology with educational objectives, ensuring modularity, and maintaining clear documentation.

Future work should focus on refining the application based on student feedback, expanding its capabilities, and exploring its potential for broader educational use.

In conclusion, the project has made a substantial contribution to cloud computing education at the university, providing a solid foundation for future advancements.

6 — Introduction

6.1 Background

This document provides an in-depth analysis of the design, development, and implementation of a new application developed for a critical component of the Cloud Computing and Cloud-based Applications course offered at the University of Groningen (RUG). As a master's level course, it plays a significant role within several academic tracks, particularly those focusing on cloud computing, distributed systems, and scalable application design. This course is also strategically positioned as a foundational prerequisite, equipping students with the necessary skills and knowledge for advanced courses in the curriculum.

The course content is divided into two main components: theoretical and practical. The theoretical component covers fundamental principles of cloud computing, including cloud service models, architecture, and security concerns. It also explores advanced topics such as cloud-native application development, microservices architecture, and continuous integration/continuous deployment (CI/CD) pipelines. This document specifically focuses on the practical component of the course, providing a detailed exploration of a target application that serves as the core project for this segment. This is designed to help students apply and implement the theoretical concepts they have learned.

The learning objectives associated with the practical component are comprehensive and aim to equip students with hands-on experience in transitioning an application from a local development environment to a fully operational cloud-based system. Specifically, students are tasked with deploying the application on the University's OpenStack cluster, a private cloud platform that provides a real-world environment for cloud computing exercises. A critical aspect of this deployment is ensuring that the application is capable of scaling to potentially serve millions of users, a requirement that underscores the importance of understanding cloud scalability and resource management.

The practical component, referred to in this document as *"the application"*, is purposefully designed with complexity to challenge students, but its modular structure ensures that it remains manageable and promotes effective learning. This modularity is crucial, as it allows for the seamless adoption of cloud technologies without necessitating modifications to the core source code. This design choice ensures that students can focus on cloud integration and deployment strategies rather than being encumbered by the need to refactor or re-engineer the application itself, at least for the first phases of the project.

The practical component is meticulously structured into several milestones, each representing a key phase in the overall project. The initial milestone involves *"dock-*

erizing" the application—encapsulating the software into Docker containers, which are then orchestrated using Docker Compose. This milestone introduces students to containerization, a fundamental skill in modern cloud computing, and emphasizes the importance of environment consistency and application portability.

Following the Docker Compose deployment, students advance to the next milestone, which involves converting the application's components into Helm charts. Helm, a package manager for Kubernetes, simplifies the deployment and management of applications within a Kubernetes cluster. This milestone not only familiarizes students with Kubernetes, an industry-standard platform for container orchestration, but also with Helm's templating and configuration management capabilities, which are essential for managing complex cloud-native applications.

The final milestone of the practical component requires students to use Terraform, an Infrastructure as Code (IaC) tool, to deploy the application on the University's OpenStack cluster. This milestone is particularly challenging as it involves establishing a highly available Kubernetes cluster across multiple virtual machines, complete with auto-scaling capabilities. This exercise is designed to simulate real-world cloud deployment scenarios, where students must manage the complexities of distributed systems, ensure fault tolerance, and optimize for performance and scalability.

This practical component, and the application at its core, represents an evolution of the course from its initial version as offered last academic year, where a different application was used. The new application is part of the University's broader initiative to enhance the Computer Science master's program at RUG, reflecting the institution's commitment to providing students with cutting-edge tools and experiences that are directly applicable to the rapidly evolving field of cloud computing.

Overall, this document not only outlines the technical aspects of the application and its role within the course but also situates it within the broader educational objectives of the Computer Science MSc programme. By providing students with a rigorous, hands-on project, the course aims to bridge the gap between theoretical knowledge and practical skills, preparing graduates to excel in both academic research and industry positions related to cloud computing and large-scale system design.

6.2 Problem Description

Problem 1 (P1) The previous application used by the course was created by a former teaching assistant (TA) to the course as a product for the *"Web and Cloud Computing"* course that served as the predecessor for the current course. However, the decision to develop the application using C# and the .NET framework introduced several challenges that ultimately affected the application's suitability for use in a course laboratory setting. Feedback from both students and TAs during the previous iteration of the course highlighted these issues, making it apparent that a complete redesign of the application was necessary to ensure it could function effectively within the course's framework. Which is the aim of this research internship.

More specifically, while the .NET framework is a robust and widely used platform, it is not extensively covered within the RUG's Computer Science curriculum, and C# is not currently part of the standard curriculum. This disconnect between the technologies used in the application and the tools and languages familiar to students seemingly presented a significant barrier to student engagement and learning. The project's reliance on .NET and C# meant that students had to invest considerable

time in learning these new technologies, detracting from their ability to focus on the core cloud computing concepts that the course aimed to teach.

Problem 2 (P2) Additionally, the project suffered from a lack of comprehensive documentation. The only guidance provided was a basic README file at the top level of the project, which offered little more than a cursory overview of the application. There were no design diagrams, no component descriptions, and no detailed explanations of the application's architecture or functionality. Moreover, critical aspects of working with .NET—such as locating environment variables, managing database connections, and handling connection strings—were not documented. This lack of documentation made it difficult for students to understand the application's inner workings, hindering their ability to extend or build upon the project, and ultimately limiting their learning experience.

Problem 3 (P3) Another significant shortcoming of the original application was the absence of a clear rationale for deploying it in a Kubernetes environment. The application did not present any obvious bottlenecks that would necessitate the use of Kubernetes, nor were there any tangible benefits of cloud infrastructure that were readily apparent to students. As a result, the value of deploying the application in a cloud-based Kubernetes environment was not evident, leading students to perceive the exercise as unnecessary and disconnected from the course's objectives. This lack of clarity around the purpose and benefits of using Kubernetes in the context of the application further undermined the effectiveness of the practical component.

Problem 4 (P4) Another issue encountered last year was the difficulty students faced in setting up the necessary technology to perform the practicals. Establishing a local Docker and Kubernetes development environment proved to be particularly challenging. Students struggled with installing and configuring the required software, and the diverse range of operating systems and hardware configurations further complicated the setup process. This difficulty significantly impeded the progress of the practical component, as many students were unable to get the environment running, leading to frustration and delays.

6.3 Envisioned solution

To summarize, it is imperative to enhance the students' experience regarding the practical component of the course. This improvement plan focuses on the following key objectives:

1. **Redesign and Rewrite the Base Application:** The base application will be rewritten and redesigned in a programming language that is more familiar to Computing Science students at the University of Groningen. The technology stack will also be updated to align with those more commonly used among the student body. Which handles P1.
2. **Provide Comprehensive Documentation:** Clear and detailed documentation will be created, outlining the technology used, the processes involved, and the interactions between different components. This will assist students in understanding the project architecture and workflow. Which handles P2.

3. **Provide a bottleneck:** The application will have a clear bottleneck that requires scaling in the cloud to lead students to perceive the exercise as necessary. Which handles P3
4. **Ensure Standardized Development Environments:** Students will be provided with ready-to-use development environments and machines (servers) on the OpenStack Clusters. This will ensure a standardized setup across all students, eliminating discrepancies in local configurations and promoting a uniform learning experience. Which handles P4

These objectives will serve as the foundation for the enhancement of the practical component of the course and will be referenced throughout the improvement plan.

In response to these challenges, and as part of this research internship, the application will not only be completely redesigned but will also be accompanied by the development of new tools aimed at creating isolated environments for each group of students on the University's OpenStack cluster. These environments, referred to as *"Virtual Labs"*, will adhere to the principles of cloud computing by providing a ready-made, cloud-hosted environment for each student group. This approach eliminates the need for local development environments, thereby resolving the issues related to software installation and configuration on personal machines. These Virtual Labs will provide a consistent and uniform environment for all students, ensuring that everyone has access to the necessary resources and tools to complete the practical component successfully.

Finally, the students encountered substantial difficulties due to the absence of necessary installations on their local machines. Many students lacked the hardware and software capabilities to run the application concurrently with a Kubernetes cluster, which is resource-intensive. These constraints prevented students from fully engaging with the practical exercises, as they were unable to simulate real-world cloud environments on their personal devices. This limitation underscored the need for a cloud-based solution that would remove these barriers and provide a more accessible, scalable environment for all students.

In conclusion, the challenges encountered with the original application—ranging from technological incompatibility to inadequate documentation and unclear objectives—highlighted the need for a comprehensive redesign. The new application and its accompanying tools, developed as part of this research internship, aim to address these issues by providing a cloud-native solution that aligns with the course's learning objectives and the technological landscape of the RUG.

7 — Methodology

7.1 Methods

To achieve these objectives, a new application will be developed using Java and the Spring Boot framework. Java is a more familiar language and is taught at the RUG in both the Object-Oriented Programming (OOP) and Advanced Object-Oriented Programming (AOOP) courses. This familiarity will enable students to understand the backend more effectively and make modifications to the code if necessary, allowing them to focus on critical aspects such as scalability and cloud integration rather than the obstacle of learning a new programming language.

To further enhance the educational value of the project, another part of the application will be written in Python, a language with which many students are also familiar. This dual-language approach will showcase the advantages of designing applications for a microservices architecture, where different components can be written in various languages or developed by separate teams. This method not only demonstrates the flexibility and modularity of microservices but also reduces confusion among students, of both components being deployed together, by clearly delineating the roles and responsibilities of each component. The process of containerizing these components using Docker and scaling them within a Kubernetes environment will be integral to the learning experience, offering students practical insights into modern cloud-based application development.

The project will be streamlined by removing certain features from the previous iteration, such as multitenancy and user authentication, as these do not contribute significantly to the course's learning objectives. Instead, the emphasis will be on core cloud computing concepts and the practical application of these concepts in a real-world scenario. To support this, a high-level diagram will be provided to give students and teaching assistants (TAs) a clear overview of the application's architecture and component interactions. Additionally, a second diagram will show how the application scales within a Kubernetes environment, with specific bottlenecks highlighted to help students understand the advantages of cloud adoption. These visual aids are intended to simplify complex ideas and enhance understanding.

7.2 Virtual Labs (VL)

To address the challenges associated with setting up local development environments, the course will implement *Virtual Labs (VL)*—isolated, cloud-based environments hosted on the University's OpenStack cluster. These Virtual Labs will provide each group of students with a pre-configured development environment that mirrors the production environment. By utilizing VLs, the need for students to in-

stall and configure complex software locally will be eliminated, thereby reducing the risk of technical issues and ensuring that all students can begin their practical work without delay.

The VLs will include all necessary tools and configurations, such as Docker, Kubernetes, and the Java and Python runtimes, pre-installed and ready to use. This approach will not only standardize the development environment across all students but will also align with the principles of cloud computing by leveraging cloud infrastructure to deliver scalable, on-demand resources. The VLs will also facilitate easier monitoring and support by TAs, as they will have access to a consistent environment when assisting students with troubleshooting and debugging.

This allows students to focus more on applying cloud computing concepts and less on overcoming technical hurdles, leading to a more effective and immersive learning experience. The use of VLs also prepares students for industry practices by familiarizing them with cloud-based workflows and infrastructure management, thus bridging the gap between academic learning and real-world applications.

In summary, the methodology adopted for this project emphasizes the use of familiar technologies, the simplification of the application architecture, and the deployment of cloud-based Virtual Labs to enhance the learning experience. By focusing on these areas, the course aims to provide students with a deeper understanding of cloud computing concepts and practical skills that are directly applicable to real-world scenarios.

8 — Results

8.1 Design

The designed application called ”Sandbox Code Runner” was meticulously engineered to ensure no coupling between the components. The application consists of three main components and three additional elements serving as data sources and a message queue. The components are decoupled in accordance with microservices architecture design guidelines, allowing each component to be scaled horizontally and independently. The main three components, depicted in Figure 8.1, are:

1. Frontend
2. Backend
3. Orchestrator

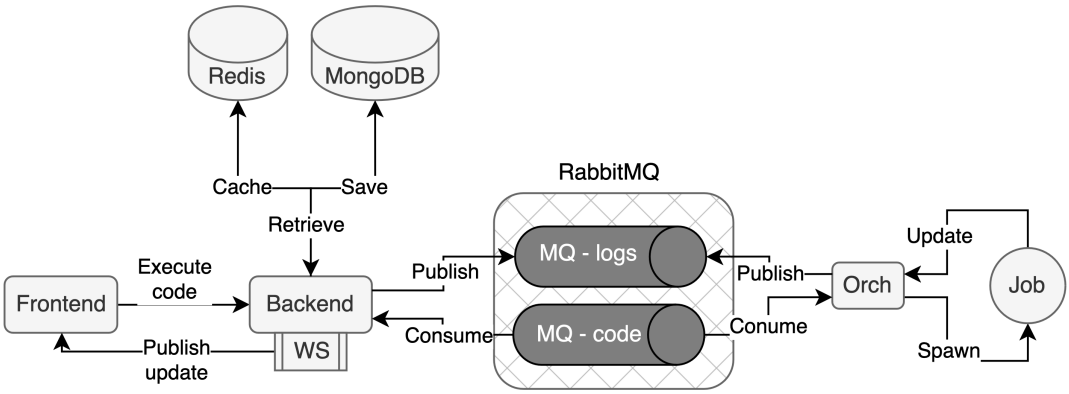

Figure 8.1: Sandbox Code Runner Architecture

8.1.1 Frontend

The frontend, developed using React JS, is responsible for handling user interactions and providing real-time feedback during code execution. It captures and processes user inputs, specifically Bash language code blocks, which are central to the system's operations. The frontend is structured around several key components such as `CodeBlockForm`, `LogsBox`, and `ExecutedBlocksList`, which collectively manage the user interface, code submission, and the display of execution results.

Before a code block is submitted, the frontend performs client-side validation to ensure the input is valid and that the backend service is available, preventing the submission of incomplete or erroneous requests. Upon successful validation, the code block is sent to the backend via a RESTful API, specifically the `/execute` endpoint.

The frontend then establishes a WebSocket connection using the `WebSocketClientContext` component, which receives real-time updates from the backend regarding the status and output of the code execution. These updates are dynamically rendered in the user interface to keep the user informed about the progress of their request.

The frontend is designed to be stateless, allowing multiple instances to be deployed behind a load balancer, which supports horizontal scaling and ensures that the system can handle a large number of concurrent users without performance degradation. This approach aligns with the overall architecture, as depicted in Figure 8.1, where the frontend communicates with the backend to publish updates and receive execution results efficiently.

8.1.2 Backend

The backend, implemented using Java and Spring Boot, serves as the core of the system's business logic and data handling. It provides several key functionalities through its RESTful API and WebSocket server. When a code block is received via the `/execute` endpoint, the backend validates the request using Data Transfer Objects (DTOs) such as `CodeBlockRequest` and `CodeBlockResponse`. This validation step ensures that the code block meets the required criteria before proceeding further.

Once validated, the backend assigns a unique ID to the code block and notifies the frontend with this ID, along with an initial WebSocket message to update the user interface. The validated code block is then forwarded to RabbitMQ using the `MessageProducer` class, where it enters the message queue system for execution.

The backend is also responsible for managing logs and metadata associated with each code block. This is handled by the `LogsManager` and `LogsMetadataManager` classes, which interface with both Redis and MongoDB for caching and persistent storage, respectively. The backend utilizes the Repository Pattern, implemented through interfaces such as `LogsRepository` and `LogsMetadataRepository`, to abstract data access and maintain clean, maintainable code. This pattern allows the backend to interact with the database layer in a consistent manner, as illustrated in Figure 8.1.

The backend's stateless architecture supports horizontal scaling, allowing it to handle increased load by deploying additional instances. This scalability is a critical aspect of the system's design, ensuring that the backend can process a large number of code execution requests efficiently.

8.1.3 Orchestrator

The orchestrator, implemented in Python, is responsible for managing the execution of code blocks within isolated Docker containers. It consumes messages from RabbitMQ, specifically from the `MQ - code` queue, as shown in Figure 8.1. The orchestrator retrieves the code block, spawns a new Docker container, and executes the code within this isolated environment.

The `orch.py` script is the core of the orchestrator, handling the lifecycle of each job from spawning the container to monitoring its execution. It interacts with the messaging system through a series of modules, including `message_handler.py` and `rabbit_config.py`, which configure the connection to RabbitMQ and manage message processing. Upon completion of the code execution, the orchestrator

updates the backend with the results, which include the exit status and any generated logs.

The results are then published to the `MQ - logs` queue, which are subsequently consumed by the backend and made available to the frontend. This flow ensures that the user receives real-time updates on the status and output of their submitted code, maintaining the responsiveness and reliability of the system.

The orchestrator's design is focused on scalability and fault tolerance. By utilizing Docker containers, the orchestrator can efficiently manage multiple concurrent jobs, with each job being isolated to prevent interference between code executions. Additionally, the use of RabbitMQ as a message broker ensures that messages are reliably delivered and processed in the correct order, contributing to the robustness of the system.

8.1.4 Design Decisions

The design of the Sandbox Code Runner was driven by the need to create a robust, scalable, and maintainable system capable of handling real-time code execution and logging in a cloud-native environment. Several key decisions were made to achieve these goals:

Message Handling in RabbitMQ

A significant design choice was the way messages are handled within RabbitMQ. Each message, representing a code block or log output, is acknowledged only after the entire execution process has been completed. This approach ensures reliable message processing and avoids the need for sticky sessions, which can complicate scalability and failover strategies. By treating each message as an atomic unit of work, the system maintains integrity even under high load, as each message is processed in its entirety before being acknowledged.

Data Caching and Persistence

The system employs both Redis and MongoDB to manage data efficiently. Logs, which represent the output of the containerized code execution, are stored in Redis for fast retrieval and in MongoDB for persistent storage. This dual storage strategy allows the system to leverage Redis's in-memory data store for quick access to frequently requested data, while MongoDB serves as the long-term data store. This design also simplifies fault tolerance, as Redis mirrors the data stored in MongoDB, ensuring that any data in MongoDB is readily available in Redis, thereby reducing latency in data access.

Fault Tolerance and Retry Mechanisms

The backend includes built-in retry mechanisms to handle transient failures, particularly in communication with RabbitMQ and the orchestrator. These retries are crucial for maintaining the system's robustness, ensuring that temporary network issues or service disruptions do not result in lost or unprocessed messages. The orchestrator, responsible for managing Docker containers that execute the code blocks, also communicates its status back to the backend. This communication includes updates on job progress and completion, which the backend then uses to update the

frontend in real-time. This ensures that the user is always informed of the current status of their code execution.

Separation of Concerns and Scalability

The architecture strictly adheres to the principles of microservices, with each component—frontend, backend, orchestrator, and message queue—being decoupled and independently scalable. This separation of concerns not only simplifies development and maintenance but also allows each component to scale independently based on demand. For instance, the frontend can scale horizontally to handle more users, while the backend and orchestrator can scale based on the number of code execution requests. RabbitMQ, as the central message broker, ensures that these components can communicate asynchronously, further enhancing the system's scalability and reliability.

Real-time Communication via WebSockets

WebSockets are used extensively to facilitate real-time communication between the backend and frontend. Unlike traditional HTTP, which follows a request-response model, WebSockets maintain a persistent connection, allowing continuous data flow. This is particularly beneficial for streaming log data from the backend to the frontend as soon as it is available. This real-time capability is essential for providing immediate feedback to users, enhancing the overall user experience.

These design decisions collectively ensure that the Sandbox Code Runner is robust, scalable, and capable of delivering a responsive user experience even under heavy load. The careful separation of concerns between different components, coupled with the use of modern technologies like RabbitMQ, Redis, MongoDB, and Docker, provides a solid foundation for the system's current functionality and future enhancements.

8.2 Technology Stack

The Sandbox Code Runner leverages a diverse and robust technology stack to ensure high performance, scalability, and ease of development. The key technologies used include:

- **Redis**: An in-memory data structure store used for caching log data, significantly reducing access times compared to querying the persistent database.
- **MongoDB**: A NoSQL database chosen for its scalability and flexibility, used to efficiently store and retrieve unstructured data.
- **Java and Spring Boot**: Java is a widely taught and robust programming language used in conjunction with the Spring Boot framework to develop the backend. Spring Boot simplifies the development of production-ready applications and provides a comprehensive suite of tools for building robust backend services.
- **Python**: Python is used for the orchestrator to demonstrate the flexibility and power of a microservices architecture, where different components can be implemented in various programming languages.

- **React JS**: React JS is employed for the frontend development. It is a popular JavaScript library for building user interfaces, known for its efficiency and flexibility in creating interactive and dynamic web applications.
- **WebSockets**: Used to facilitate real-time, bidirectional communication between the frontend and backend, maintaining a persistent connection for continuous data flow, which is crucial for real-time updates.
- **RabbitMQ**: A message broker that manages the communication between the backend and the orchestrator, allowing for distributed processing and task management. It is essential for handling the asynchronous nature of the system's tasks.
- **Docker**: Containerization technology used to package applications and their dependencies into containers, ensuring consistency across different environments and enabling scalable deployments.
- **Kubernetes**: Orchestrates the deployment, scaling, and operation of containerized applications, managing the backend, frontend, and orchestrator components to ensure the system can scale horizontally based on demand.
- **Ansible**: An automation tool used for provisioning and configuration management. Ansible ensures that the infrastructure is consistently deployed and configured across different environments, streamlining the process of scaling the system and maintaining its reliability.
- **Terraform**: An Infrastructure as Code (IaC) tool used to provision and manage cloud resources efficiently. Terraform enables the automated deployment of infrastructure components, ensuring consistency and scalability across different environments.
- **GitLab CI/CD**: GitLab's Continuous Integration/Continuous Deployment (CI/CD) pipelines are used to automate the building, testing, and deployment of the application. This integration ensures that new code changes are automatically tested and deployed, maintaining high code quality and enabling rapid iteration.

This technology stack provides a solid foundation for the system, enabling it to be scalable, resilient, and responsive to user demands, while also ensuring that the infrastructure and deployment processes are automated and efficient.

8.3 Scalability

The scalability of this system is achieved through a combination of architectural design choices, component decoupling, and the utilization of distributed technologies. Each component in the system—from the frontend to the backend, RabbitMQ, Redis, and MongoDB—is selected and configured to support horizontal scaling, ensuring that the system can handle increased load and expand its capacity as demand grows. This scalability is managed and facilitated by Kubernetes for orchestration and Ansible for provisioning and configuration management, which ensures consistent and automated deployments across different environments.

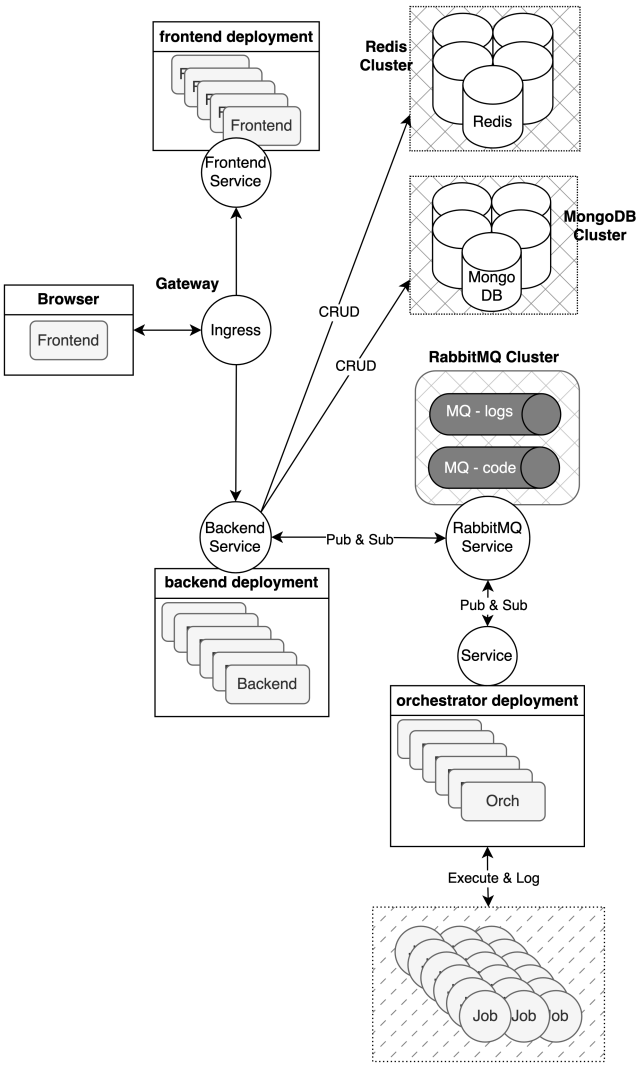

Figure 8.2: Scaling Diagram

8.3.1 Frontend Scalability

The frontend is inherently stateless, which allows for easy horizontal scaling. Multiple instances of the frontend can be deployed behind a load balancer, distributing incoming user requests across these instances. This design ensures that the system can support numerous simultaneous users without experiencing performance degradation. The use of WebSockets, which typically presents scaling challenges due to the persistent nature of connections, is managed through Kubernetes Horizontal Pod Autoscaler (HPA) configurations defined in the Helm charts. This configuration allows the frontend deployment to scale based on CPU or memory usage, dynamically adjusting the number of running instances to match the current load.

8.3.2 Backend Scalability

The backend is designed to be stateless and modular, which is critical for scaling. The stateless nature of the backend means that it can easily scale horizontally by adding more instances to handle increased requests. Docker is used to containerize the backend, and these containers are orchestrated using Kubernetes. The scaling

process is automatically managed by Kubernetes HPA. Ensuring that the backend deployment can handle fluctuations in demand without manual intervention.

The backend's reliance on RabbitMQ for handling code execution and log messages allows for distributed processing. RabbitMQ supports clustering, which means that the message queue itself can be scaled horizontally. This clustering is configured, allowing RabbitMQ to distribute messages across multiple nodes, which prevents any single node from becoming a bottleneck. Worker processes, in the form of backend instances, can then consume these messages concurrently, enhancing the system's ability to process a large volume of execution requests and log processing tasks simultaneously.

8.3.3 Message Queue Scalability

RabbitMQ is central to the scalability of this system. It acts as a broker for distributing tasks and log messages, enabling asynchronous processing and decoupling between the backend and the orchestrator. RabbitMQ's clustering capability is a key feature that supports its scalability, allowing multiple RabbitMQ nodes to work together, share the load, and provide fault tolerance. This configuration ensures that if one node fails, another can take over without disrupting the entire system.

Moreover, RabbitMQ supports exchange and queue federation, which enhances its scalability by allowing federated exchanges and queues to span multiple RabbitMQ clusters. This setup can be particularly useful in geographically distributed environments, ensuring low-latency message delivery and high availability even in large-scale systems.

8.3.4 Data Storage Scalability

Both Redis and MongoDB are chosen for their robust scalability features. Redis, acting as a cache layer, can be scaled horizontally by adding more nodes to a Redis cluster. This allows Redis to handle more read and write operations concurrently, which is crucial as the number of concurrent users and the volume of logs increase.

MongoDB, used for persistent storage, is a NoSQL database known for its scalability. MongoDB supports sharding, a process that distributes data across multiple servers, enabling the database to handle large datasets and high throughput. The sharding configuration, managed via Kubernetes, ensures that MongoDB can scale horizontally by adding more shards as the dataset grows. Additionally, MongoDB's replica sets distribute read operations across multiple nodes, reducing the load on any single node and contributing to overall system performance.

8.3.5 Orchestrator and Job Management Scalability

The orchestrator component, which manages the execution of code blocks in isolated Docker containers, is designed to scale with the number of incoming jobs. Kubernetes is used to dynamically allocate resources for job execution based on current demand, with HPA configurations automatically adjusting the number of orchestrator instances. This dynamic scaling capability is essential for maintaining performance and reliability, especially when handling numerous concurrent jobs.

8.3.6 System Scalability

In conclusion, the system's scalability is underpinned by its modular, decoupled architecture and the strategic use of technologies that support horizontal scaling and high availability. The use of Kubernetes for orchestration, combined with Ansible for provisioning, ensures that the system can dynamically adjust to changes in load and maintain performance under varying conditions. By distributing workloads across multiple instances, clusters, and nodes, the system can accommodate increasing user demands and data volumes without compromising performance or reliability. This scalability ensures that the system remains robust and responsive, even as it grows in size and complexity.

8.4 Infrastructure Automation and Deployment Workflow

The system's infrastructure and deployment processes are fully automated using a combination of Terraform, Ansible, and GitLab CI/CD. This integrated approach ensures that infrastructure is consistently provisioned, configured, and maintained across different environments, supporting the system's scalability, reliability, and operational efficiency.

8.4.1 Infrastructure as Code (IaC) with Terraform

Terraform is employed to define and provision the cloud infrastructure for the system using Infrastructure as Code (IaC) principles. This enables the infrastructure setup to be described in code, ensuring repeatable, consistent, and auditable deployments. Key components such as virtual machines, networking configurations, and security groups are defined within Terraform configuration files.

The state of the infrastructure, managed by Terraform, is securely stored and versioned in GitLab, allowing for collaborative infrastructure management and preventing conflicts. This setup is crucial for maintaining an accurate and reliable representation of the infrastructure across all environments.

8.4.2 Configuration Management with Ansible

Once the infrastructure is provisioned by Terraform, Ansible automates the configuration of servers and services. Ansible playbooks are executed to install necessary software, configure system settings, and deploy applications across the infrastructure. This automation ensures consistency across all servers, reducing the potential for configuration drift and minimizing manual intervention.

The playbooks are triggered as soon as the Master VM is provisioned, pulling the latest resources from GitLab, installing required packages such as Docker, and applying Helm charts to set up the environment. Ansible integrates seamlessly with GitLab CI/CD pipelines, ensuring that any configuration changes are automatically tested and deployed, maintaining a consistent and stable environment.

8.4.3 Continuous Integration and Deployment with GitLab CI/CD

GitLab CI/CD pipelines automate the deployment process, orchestrating the execution of Terraform and Ansible scripts whenever changes are made to the infrastructure or application code. These pipelines ensure that the system remains up to date with the latest configurations, applying changes in a controlled and reliable manner.

The CI/CD pipelines handle the entire lifecycle of the deployment, from building Docker images, pushing them to a container registry, to deploying the updated images on the OpenStack cluster. This approach ensures continuous integration and deployment, minimizing downtime and enhancing the reliability of the system.

8.4.4 Deployment Workflow and Environment Setup

The deployment workflow integrates multiple tools and repositories to effectively manage both the infrastructure and application components. The project is organized into several GitLab repositories, each corresponding to different system components such as the frontend, backend, orchestrator, and infrastructure resources. Each repository is equipped with its own CI/CD pipeline, automating the build, test, and deployment processes.

As illustrated in Figure 8.3, when a change is pushed to any of these repositories, the respective CI/CD pipeline is triggered. The pipeline builds Docker images, pushes them to a container registry, and deploys the updated images to the OpenStack cluster using Ansible scripts.

8.4.5 OpenStack Cluster Configuration

The application is hosted on an OpenStack cluster comprising a master VM and two worker VMs. Terraform scripts are used to provision this infrastructure, including the setup of networks, VM instances, and security groups. Once the VMs are up and running, Ansible playbooks configure the Kubernetes cluster on these VMs, which manages the deployment of the application containers.

The master VM orchestrates the Kubernetes cluster, while the worker VMs handle the application workloads. This configuration ensures that the system is scalable, resilient, and can be centrally managed through Kubernetes.

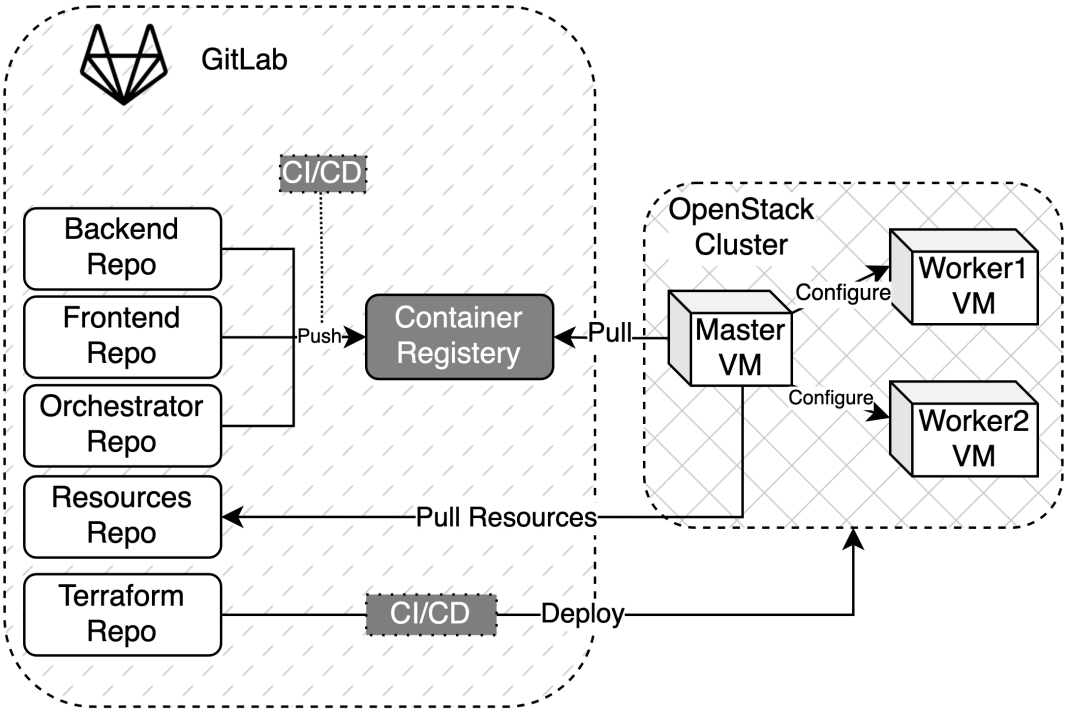

Figure 8.3: Deployed Application Overview

8.5 Virtual Labs

The Virtual Labs (VL) implementation is designed to streamline the deployment of isolated, cloud-based development environments for student groups, leveraging the University's OpenStack cluster. This approach ensures that each group of students has access to a consistent and fully configured environment, eliminating the need for local software installation and configuration, and aligning with the cloud computing principles taught in the course.

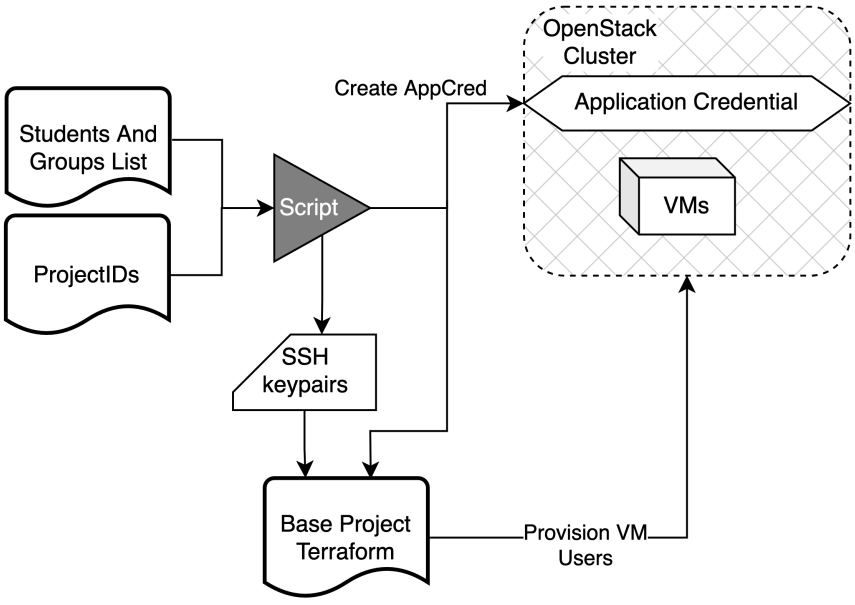

Figure 8.4: Virtual Labs

As depicted in Figure 8.4, the process begins with the provision of a CSV file that contains details about the students and their respective groups. This file is crucial as it serves as the basis for the subsequent steps in the Virtual Labs setup. The following steps outline the detailed process:

1. **Project Creation on OpenStack:** For each group listed in the CSV file, a dedicated project is created on the OpenStack cluster. This project acts as an isolated environment where the group can conduct their development activities. Each project is uniquely mapped to a group and assigned a corresponding project ID. This mapping ensures that the resources and environments are properly segregated and managed for each group, maintaining isolation and preventing cross-group interference.

2. **User SSH Key Pair Generation:** For each student, an SSH key pair is generated, with the public key being used to configure secure SSH access to the VM.

3. **Application Credentials (App Creds) Generation:** After the project creation, App Creds are generated for each project. These credentials are essential as they are the only possibility of interacting with the OpenStack API and resources. The generated App Creds allow for programmatic access to the project, enabling automated deployment and management of the environment via Terraform.

4. **Terraform-Based Deployment:** With the App Creds in place, the base Terraform project is deployed within each group's project on the OpenStack cluster. Terraform workspaces are utilized to manage the different group environments, allowing for scalable and repeatable infrastructure deployment. During this step, the necessary software is installed on the virtual machines (VMs) provisioned for each group. This software typically includes development tools such as Docker, Kubernetes, and the Java and Python runtimes, pre-configured to match the course requirements.

5. **User Account and SSH Key Pair Generation:** As the VMs are being created, individual user accounts are set up for each member of the group. This ensures that each student can securely access their group's environment using their personal credentials. The SSH key pair, along with the student's unique username (typically their student number, referred to as *snumber*), is critical for maintaining secure and personalized access to the cloud environment.

6. **Provisioning Access Details to Students:** Once the VM setup is complete, each student is provided with the necessary access details. This includes the floating IP address of their group's VM, their individual username, and their SSH key pair. These details are crucial for enabling students to connect to their Virtual Lab environment and begin their work. By providing these credentials directly, the process ensures that students can immediately start their practical exercises without the need for additional setup or troubleshooting, fostering a seamless transition to the cloud-based development environment.

This approach to Virtual Labs not only simplifies the technical setup for students but also enhances the scalability and manageability of the course's practical component. By automating the deployment process through Terraform and integrating secure access mechanisms, the course ensures that each student group has a robust and consistent environment tailored to their learning needs. The use of OpenStack's cloud

infrastructure further aligns the practical experience with real-world cloud computing practices, providing students with valuable hands-on experience in managing and deploying cloud-based applications.

9 — Discussion

9.1 Summary of Key Findings

The redesign and implementation of the new application for the "Cloud Computing and Cloud-based Applications" course yield several significant improvements that enhance both the educational and technical aspects of the course. These enhancements are carefully aligned with the course's objectives, preparing students more effectively for real-world cloud computing challenges.

One of the most impactful outcomes is the overall increase in student engagement and participation. This is achieved through several key innovations, including the introduction of Virtual Labs (VLs), the adoption of a modern and familiar technology stack, and the automation of infrastructure management. Together, these elements reduce the technical barriers that previously hindered students, allowing them to focus more fully on the core learning objectives.

Additionally, the uniform environment provided by the VLs enhances collaboration and teamwork among student groups. With all students operating within the same setup, troubleshooting and knowledge sharing become more straightforward, promoting a cohesive and productive learning environment. This consistency supports the development of collaborative skills that are essential in both academic and professional contexts.

The application leverages a technology stack centered around Java, Spring Boot, and Python—tools that are both widely taught in the academic setting and extensively used in the industry. This approach not only makes the application more accessible but also provides students with relevant, hands-on experience. The modular microservices architecture introduces them to essential concepts of cloud computing.

Infrastructure automation is another key area of focus. By integrating Infrastructure as Code (IaC) using Terraform and automating configurations with Ansible, the course introduces students to advanced DevOps practices critical for managing modern cloud infrastructures. These tools streamline the deployment process and provide students with practical experience in efficiently managing and scaling complex systems. The expectation is that students will make the application capable of potentially handling millions of users, reflecting real-world demands for scalable systems.

The architectural design of the application, emphasizing decoupled components and efficient message handling through RabbitMQ, results in a robust and reliable system. This design not only supports the educational objective of teaching scalability but also ensures that students understand how to build systems that are both fault-tolerant and capable of scaling horizontally, a crucial aspect of cloud-native applica-

tions.

In summary, the project successfully addresses the limitations of the previous system by introducing a range of technical and educational enhancements. These improvements not only meet the project's objectives but also significantly enrich the learning experience, equipping students with the skills and knowledge necessary to navigate and succeed in the complexities of cloud computing.

9.2 Challenges

The development of the new application was a significant undertaking, particularly as it was driven by a single student as part of a research internship. The process involved not just a change in the technology stack but a complete rethinking of the application's concept and structure.

One of the key challenges was the complete overhaul of the existing application. The previous system, based on .NET and C#, was not well-aligned with the course's educational goals or the students' technical backgrounds. The decision to shift to a new technology stack—centered around Java, Spring Boot, and Python—required a re-envisioning of the entire application. This was not just a technical upgrade but also a conceptual redesign, ensuring that the new application better served the course's focus on cloud computing and scalability.

Redesigning the application's architecture to meet these new goals involved careful planning. The new system needed to be modular and scalable, supporting the educational objective of teaching students about cloud scalability. This meant designing an application skeleton that students could extend to potentially handle millions of users, thereby giving them practical experience in building scalable solutions.

The introduction of VLs was another pivotal aspect of the redesign. These labs were intended to provide students with consistent, pre-configured cloud environments, which would eliminate many of the technical barriers encountered in previous iterations of the course. Developing these labs required addressing various technical issues related to cloud resource management, security, and network configurations. Ensuring that the VLs were both robust and easy for students to use was a significant part of the development process.

Managing the entire project as a solo developer required a high level of self-discipline and problem-solving ability. Balancing the need to redesign the application's concept, implement a new technology stack, and ensure the system's scalability and usability within the constraints of the project timeline was no small task. However, these challenges were met with a focus on delivering a functional, educationally effective tool that aligned with the course's objectives.

In summary, the development of the new application involved navigating several complex challenges, from conceptual redesign to technical implementation. The successful outcome of the project demonstrates the ability to effectively manage and overcome these challenges, resulting in a robust and innovative application that significantly enhances the learning experience.

9.3 Practical Implications

The redesigned application delivers substantial practical benefits, directly enhancing students' readiness for industry challenges. By incorporating industry-standard tools like Java, Spring Boot, Python, Kubernetes, and Terraform, the course now provides students with hands-on experience that closely mirrors real-world practices. This exposure not only bridges the gap between theoretical knowledge and application but also equips students with skills that are highly valued in the job market.

The introduction of VLs ensures that all students have equal access to a consistent, cloud-based environment, significantly reducing technical barriers and allowing them to focus on mastering cloud computing concepts. This uniformity fosters a more inclusive learning experience, making the course accessible to a wider range of students, regardless of their prior technical background.

A key focus on scalability within the course equips students with the ability to design and implement systems capable of handling large-scale operations, an essential skill in cloud computing. The requirement for students to extend the application to handle potentially millions of users instills a deep understanding of scalability, preparing them to tackle similar challenges in professional settings.

The integration of infrastructure automation tools like Terraform and Ansible introduces students to modern DevOps practices, providing them with practical experience in managing and automating cloud infrastructures. This knowledge is crucial for operating in environments where continuous integration and deployment are standard.

Moreover, the standardized setups facilitated by VLs promote effective teamwork and collaboration, essential skills in the software development industry. The course structure encourages students to work together within a consistent environment, enhancing both their technical and soft skills.

In summary, this project not only enhances the educational value of the course but also significantly boosts students' practical capabilities, better preparing them for careers in cloud computing and related fields.

9.4 Lessons Learned

Throughout the development of the new application, several key lessons emerged that are valuable for both future projects and educational design.

The decision to structure the application around a microservices architecture underlined the value of modularity. This approach not only made the application scalable, but also provided a clear, tangible way for students to interact with and learn about complex cloud systems. The modular design facilitated incremental learning, allowing students to build and extend the application in manageable steps, reinforcing their understanding of key cloud principles.

The introduction of VLs highlighted the critical impact of providing students with a consistent, ready-to-use environment. By eliminating setup challenges, the VLs allowed students to focus more on learning and experimentation, which led to better engagement and outcomes. This lesson reinforces the importance of simplifying

technical environments in educational settings to allow for a more focused and productive learning experience.

One lesson learned was the critical role of comprehensive and clear documentation in the success of the project. Detailed documentation not only supported the development process but also ensured that students could navigate the application and its components effectively. This experience highlighted that robust documentation is essential, especially in educational tools, as it helps bridge the gap between complex technology and the learner's understanding.

Finally, the project revealed the delicate balance required between introducing advanced technical concepts and maintaining accessibility for all students. Ensuring that the application was challenging enough to be instructive, yet approachable for those with varying levels of experience, was a key factor in its success. This balance is essential in educational design, where the goal is to push students' understanding without overwhelming them.

9.5 Recommendations for Future Research

While the new application has significantly enhanced the learning experience in the "Cloud Computing and Cloud-based Applications" course, there are several avenues for future research that could further improve its effectiveness and extend its impact.

Given the need to finalize the application before the course commenced, for example, there is significant potential for refining the application based on student feedback gathered during and after its initial use. Future research could focus on systematically collecting and analyzing this feedback to identify areas where the application can be improved. This might involve tweaking the user interface for better usability, adjusting the complexity of certain tasks to better match student skill levels, or enhancing the clarity of instructional materials. Incorporating real-world feedback from students who have used the application in a live educational setting will be crucial in making iterative improvements that enhance its effectiveness and accessibility.

In conclusion, the new application provides a solid foundation for cloud computing education, but there is potential for future research to expand its capabilities, enhance its impact, and adapt its principles to other areas of study.

10 — Conclusion

This paper detailed the redesign and implementation of a new application for the ”Cloud Computing and Cloud-based Applications” course at the University of Groningen. The project successfully replaced the previous application with a modern, scalable one that aligns with current industry standards and educational goals.

Key innovations included the adoption of a microservices architecture and the introduction of Virtual Labs, which significantly enhanced the learning experience by providing a consistent, cloud-based environment for students. These improvements facilitate better engagement, practical learning, and collaboration among students.

The development process, managed by the author of this paper, involved overcoming several technical and conceptual challenges, ultimately resulting in a robust and effective educational tool. Lessons learned from this project emphasize the importance of aligning technology with educational objectives, ensuring modularity, and maintaining clear documentation.

Future work should focus on refining the application based on student feedback, expanding its capabilities, and exploring its potential for broader educational use.

In conclusion, the project has made a substantial contribution to cloud computing education at the university, providing a solid foundation for future advancements.