
Major Project Report

CS 816: Software Production Engineering

# Peer Grading using a DevOps Toolchain

H Anarghya
MT2023083

Vedantee Pathak
MT2023123

GUIDED BY
Prof B. Thangaraju

## Preface

This is a report of the major project carried out as a part of coursework in `CS 816: Software Production Engineering`, taught by Prof B. Thangaraju at IIIT Bangalore during Spring 2024. As organizations seek to rapidly deliver software products and services to meet evolving business needs, DevOps has emerged as a critical practice for achieving this goal. DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to enable organizations to deliver software products and services more quickly, reliably, and with higher quality. This report is a documentation of a project on DevOps, which aims to explore the benefits and challenges of adopting DevOps practices in an organization. The project involved the implementation of a DevOps pipeline for a simple application for peer grading, using popular tools and technologies such as Jenkins, Docker, Kubernetes, and Git, among others.


## Acknowledgements

We would like to express our sincere gratitude to Prof. B Thangaraju and our TA Siva Jagadesh for their guidance and support throughout the course and this project on DevOps. Their expertise and insights have been invaluable in shaping our understanding of DevOps and its practices.

We would like to thank our professor for designing and teaching this course, which provided us with a comprehensive understanding of the principles and practices of DevOps. Their lectures, assignments, and feedback have been instrumental in our learning journey.

We would also like to thank our TA Siva Jagadesh for their tireless efforts in providing us with assistance and support throughout the course and this project. Their feedback, suggestions, and guidance have been critical in helping us implement the DevOps pipeline and overcome the challenges we faced.

Lastly, the group members would like to thank each other for all their support, motivation and constant backing.

# Contents

# 1 Introduction to the Application

We create an online web platform,**PeerPulse** designed to facilitate peer feedback and grading sessions in a classroom setting. The platform offers a number of features geared towards streamlining the feedback process, which is traditionally a logistically overwhelming task that is typically done manually or via tedious forms.

One key feature of the website is the ability for instructors to create courses and form groups for each course. Students can then sign up using a course code or an invite link from the platform. This allows for easy management of multiple courses and groups, and ensures that only registered students are able to access the feedback system.

Another important feature of the platform is the ability for instructors to publish activities and topics from their end which the students can then respond to. These activities can include a variety of different types of questions, such as scoring the contributions of other group members, writing feedback for them, reflecting on everyone's contribution, and more. This allows for a flexible feedback process that can be tailored to the needs of each individual course.

The platform also offers the ability to generate a score based on rubrics defined by the instructor. This score can be used as a grading component for the course, saving instructors a significant amount of time and effort in the grading process.

In addition to these key features, the platform also offers a number of other benefits. For instance, the feedback system allows for flexible feedback paths between teams, from instructors to students, and from each student to other students. Powerful visibility control limits who can see the response text, the identity of the feedback giver, and the identity of the feedback receiver. The platform also offers the ability to generate exhaustive reports and statistics, allowing instructors to track student progress over time.

Finally, the platform offers a wide range of question types, including essay questions, MCQ questions, multiple select questions, numeric scale questions, and questions measuring contribution in team projects. This ensures that the feedback system is flexible enough to accommodate a wide range of different types of courses and assignments, making it a valuable tool for instructors across a range of different disciplines.

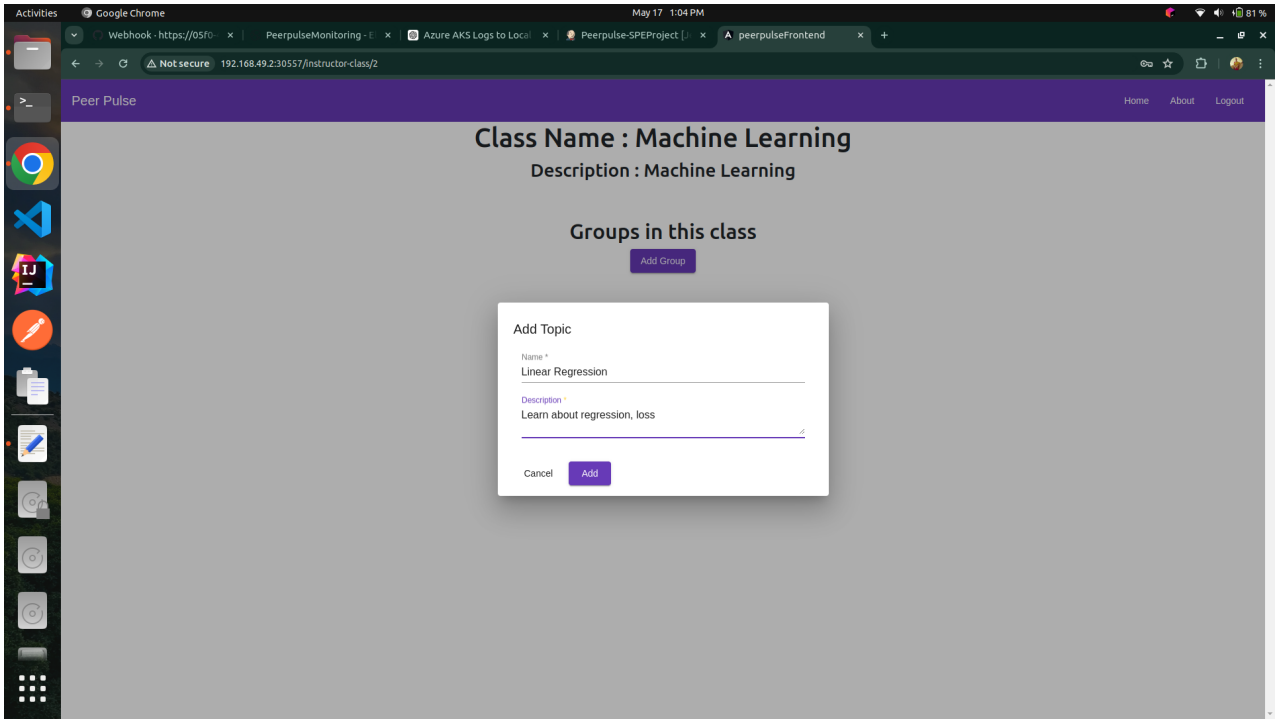



Figure 1: A snapshot from the application

### 1.1 System Architecture

The system architecture of the web application built using Angular, Spring Boot, and MySQL involves a three-tier architecture, consisting of the presentation layer, application layer, and data layer. These layers are described as:

- **Presentation Layer**: The presentation layer is responsible for displaying the user interface and handling user interactions. In this case, Angular is used to build the frontend of the web application. It runs in the user's web browser and communicates with the backend through RESTful APIs.
- **Application Layer**: The application layer is responsible for handling business logic and processing user requests. In this case, Spring Boot is used to build the backend of the web application. It provides a range of features for building RESTful APIs, handling requests, and interacting with the database.
- **Data Layer**: The data layer is responsible for storing and retrieving data. In this case, MySQL is used as the database management system. It stores the application data and provides a range of features for querying and manipulating that data.

In this architecture, the Angular frontend communicates with the Spring Boot backend through RESTful APIs. The backend processes user requests and interacts with the MySQL database to retrieve or store data. The architecture is designed to be scalable, reliable, and maintainable, with each layer responsible for a specific set of tasks.

The three-tier architecture of this application built using Angular, Spring Boot, and MySQL provides a flexible and scalable solution for building modern web applications. By leveraging the strengths of each technology, developers can create powerful and user-friendly applications that are capable of handling complex business logic and large amounts of data.

## 2 Technology Stack Used

The tech stack of an application typically refers to the set of technologies and tools used to develop and deploy the application. In this case, the tech stack includes Angular for the frontend, Spring Boot for the backend, MySQL for the database, Git and GitHub for version control, Docker for containerization, Kubernetes for container orchestration, and Azure for cloud deployment.

**Angular** is a popular frontend framework that enables developers to build dynamic, single-page web applications. It provides a range of features and tools for building responsive, scalable, and user-friendly applications.

**Spring Boot** is a powerful backend framework that simplifies the process of building and deploying Java-based applications. It provides a range of features and tools for building RESTful APIs, managing dependencies, and integrating with databases.


**MySQL** is a widely used open-source relational database management system that provides a powerful and scalable solution for storing and managing data.


**Git** and **GitHub** are popular version control systems that enable developers to track changes to their codebase, collaborate with others, and manage different versions of their code.


**Docker** is a containerization platform that enables developers to package their applications and dependencies into lightweight, portable containers. This makes it easier to deploy and run applications across different environments and operating systems.


**Kubernetes** is a container orchestration platform that enables developers to manage and scale their containerized applications more effectively. It provides a range of features for automating deployment, scaling, and management of containerized applications.

**Azure** is a popular cloud computing platform that provide a range of services and tools for deploying and managing applications in the cloud, providing scalable, reliable, and cost-effective solutions for hosting applications, managing resources, and delivering services to users.


This tech stack provides a powerful set of tools and technologies for building and deploying modern web applications. By leveraging these technologies, developers can build scalable, reliable, and user-friendly applications that can be deployed and managed more effectively in the cloud.

## 3 What is DevOps?

DevOps is a software development approach that emphasizes collaboration, communication, and integration between development and operations teams. The goal of DevOps is to enable organizations to deliver high-quality software at a faster pace, with more reliability and stability.

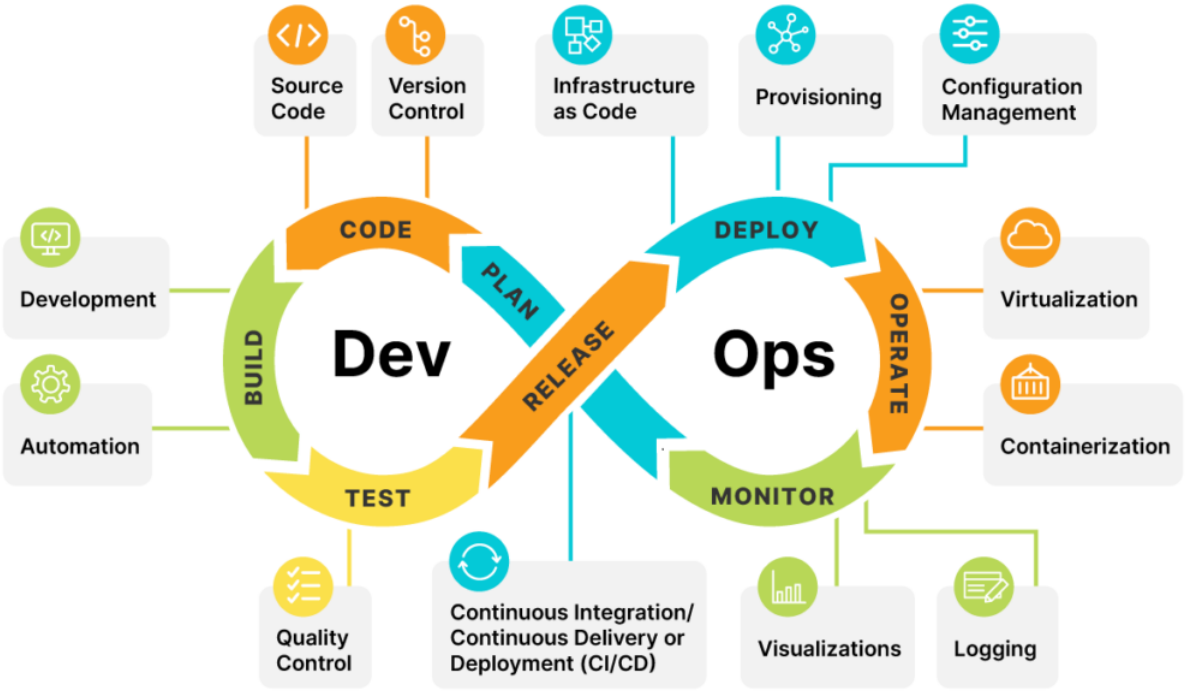



Figure 2: The DevOps scheme of things

Traditionally, software development involved separate teams for writing code and managing infrastructure. The code would be handed off to the operations team to deploy and maintain, which often led to communication breakdowns and delays in software delivery. DevOps seeks to overcome these challenges by fostering a culture of collaboration and continuous improvement between development and operations teams.

At its core, DevOps is about breaking down the barriers between development and operations teams and encouraging them to work together towards a common goal. This involves adopting a number of different practices, tools, and methodologies that are designed to facilitate communication and collaboration between these teams.

One of the key practices of DevOps is continuous integration and continuous delivery (CI/CD). CI/CD involves automating the process of building, testing, and deploying software, so that changes can be quickly and easily integrated into the production environment. By automating these processes, organizations can deliver software updates more frequently and with greater consistency, while reducing the risk of errors or downtime.

Another important aspect of DevOps is infrastructure as code (IaC). IaC involves using code to automate the provisioning and management of infrastructure resources, such as servers, databases, and networks. By treating infrastructure as code, organizations can easily scale their infrastructure up or down as needed, while also ensuring that their infrastructure is consistent and repeatable.

DevOps also emphasizes the importance of monitoring and logging. By closely monitoring the performance of their software and infrastructure, organizations can quickly identify and address issues before they become critical. This helps to ensure that their systems remain reliable and stable over time.

Finally, DevOps is about creating a culture of continuous improvement. This involves encouraging teams to regularly reflect on their processes and practices, and to identify areas where they can improve. By continuously iterating and improving their processes, teams can increase their efficiency, reduce their risk, and deliver higher-quality software.

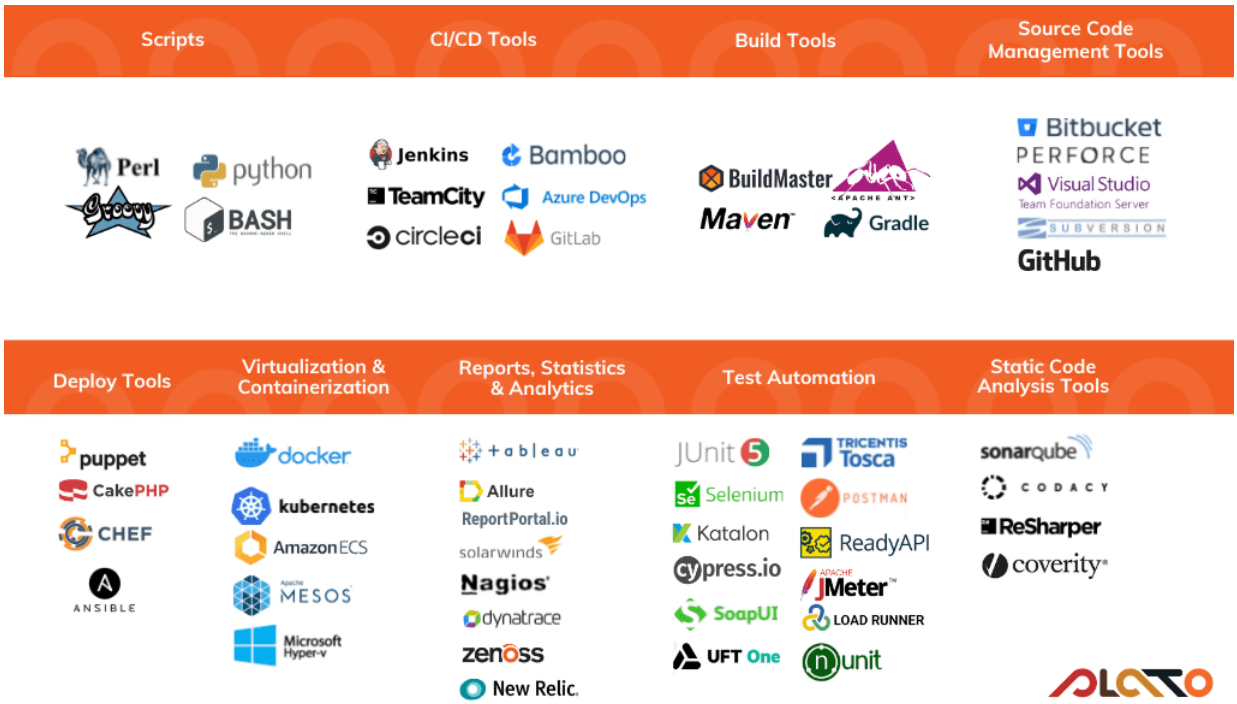



Figure 3: Some popular tools used in DevOps

DevOps is a powerful approach to software development that emphasizes collaboration, communication, and integration between development and operations teams. By adopting DevOps practices and methodologies, organizations can deliver high-quality software at a faster pace, with more reliability and stability, while also fostering a culture of continuous improvement.

## 4 Why use DevOps?

DevOps is a software development approach that has gained popularity in recent years due to its many benefits. We will detail some of the key reasons why organizations should consider adopting DevOps practices.

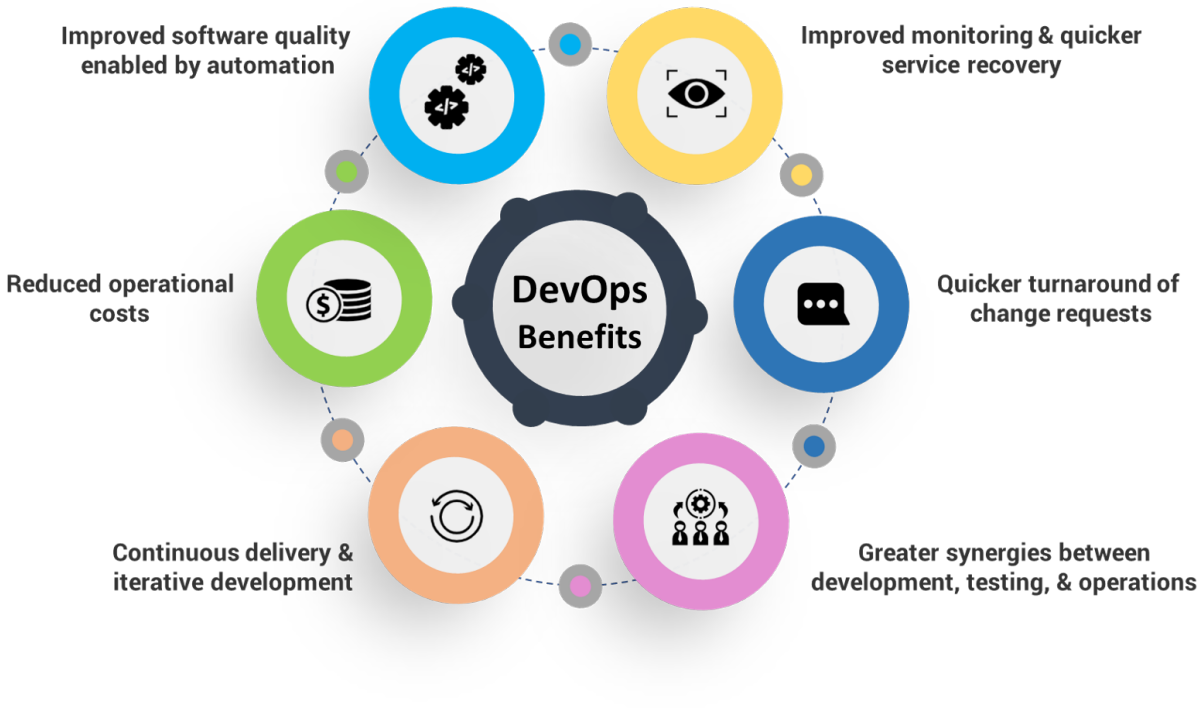



Figure 4: Benefits of DevOps

- **Faster Time to Market**: One of the primary benefits of DevOps is that it enables organizations to deliver software updates more quickly. DevOps practices such as continuous integration and continuous delivery (CI/CD) help teams to automate the process of building, testing, and deploying software, which reduces the time it takes to get new features or bug fixes into the hands of users. By releasing software updates more frequently, organizations can stay ahead of the competition and better meet the needs of their customers.
- **Improved Quality**: DevOps practices can also help to improve the quality of software. By automating the testing process, teams can catch bugs or issues earlier in the development cycle, which reduces the risk of releasing software with critical bugs or security vulnerabilities. Additionally, by adopting a culture of continuous improvement, teams can identify and address issues more quickly, which helps to keep software quality high over time.
- **Increased Collaboration**: DevOps emphasizes collaboration between development and operations teams. By breaking down silos and encouraging teams to work together, organizations can improve communication and reduce the risk of miscommunication or misunderstandings. This leads to more effective collaboration, which can help teams to identify and resolve issues more quickly.
- **Greater Agility**: DevOps practices can also make organizations more agile. By automating processes and breaking down barriers between teams, organizations can respond more quickly to changing market conditions or customer needs. This enables teams to pivot more easily and adapt to new challenges or opportunities, which can be critical in today's rapidly evolving business environment.
- **Improved Security**: DevOps practices can also help to improve the security of software. By automating security testing and integrating security into the development process, teams can catch security vulnerabilities earlier in the development cycle, which reduces

the risk of critical security breaches. Additionally, by adopting a culture of continuous improvement, teams can identify and address security issues more quickly, which helps to keep software secure over time.

- **Increased Efficiency**: DevOps practices can also help organizations to become more efficient. By automating processes and reducing manual interventions, teams can reduce the amount of time and effort required to deliver software. This can help to free up resources and reduce costs, which can be especially important for smaller organizations or those with limited resources.

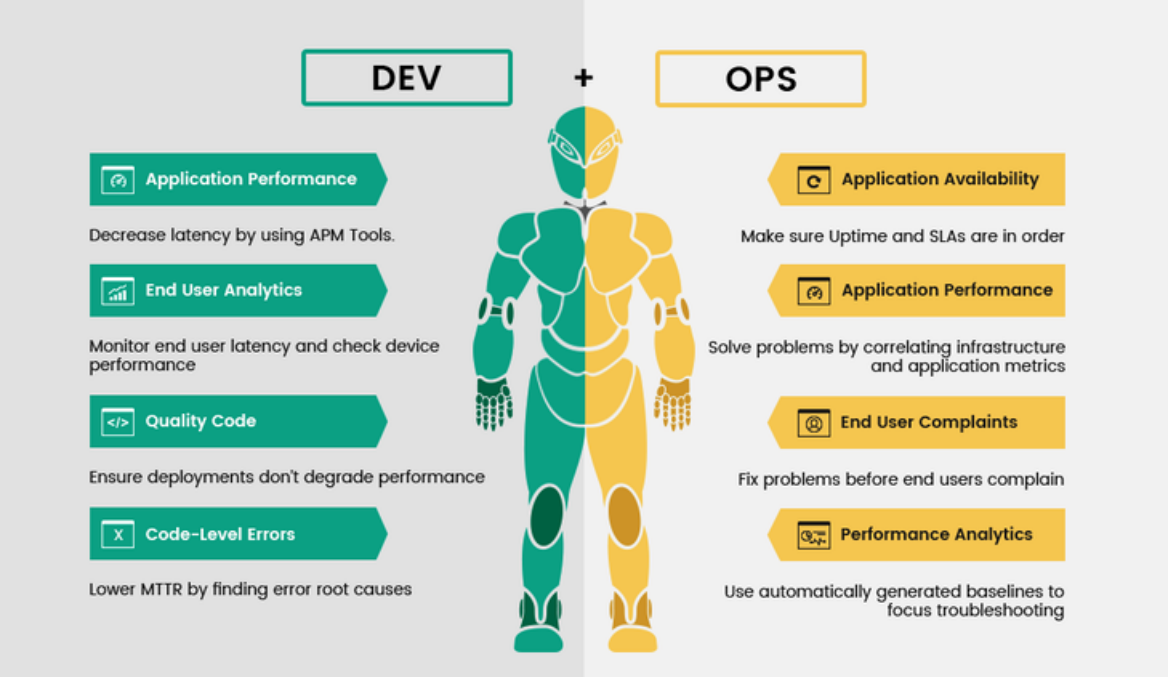



Figure 5: Interplay between the Development and Operations teams

The benefits of DevOps are numerous and significant. By adopting DevOps practices, organizations can deliver software more quickly, improve software quality, increase collaboration, become more agile, improve security, and increase efficiency. These benefits can help organizations to stay ahead of the competition, better meet the needs of their customers, and drive business success over the long term.

# 5 System Configuration

## 5.1 Local (for testing)

The application development was performed on a MacBook Pro Intel with the following system configuration: 4-core Intel CPU, 8GB Memory, and 256GB SSD storage. The operating system used was macOS Big Sur version 11.2.3.

## 5.2 Azure and Azure Kubernetes Service (for cloud deployment)

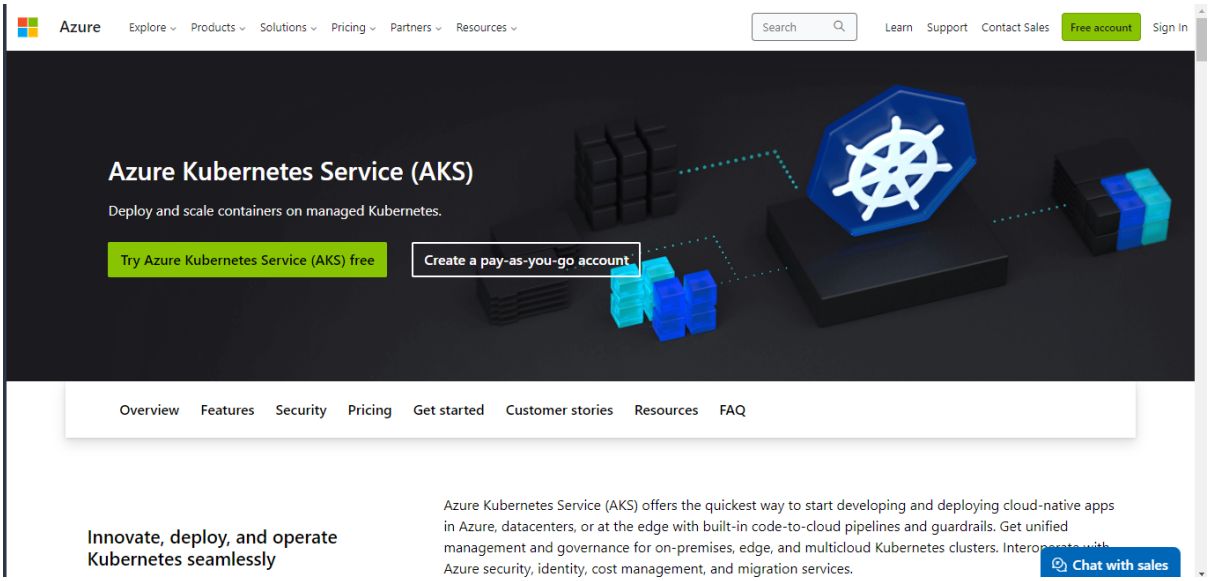



Figure 6: A description of the Azure Kubernetes Service

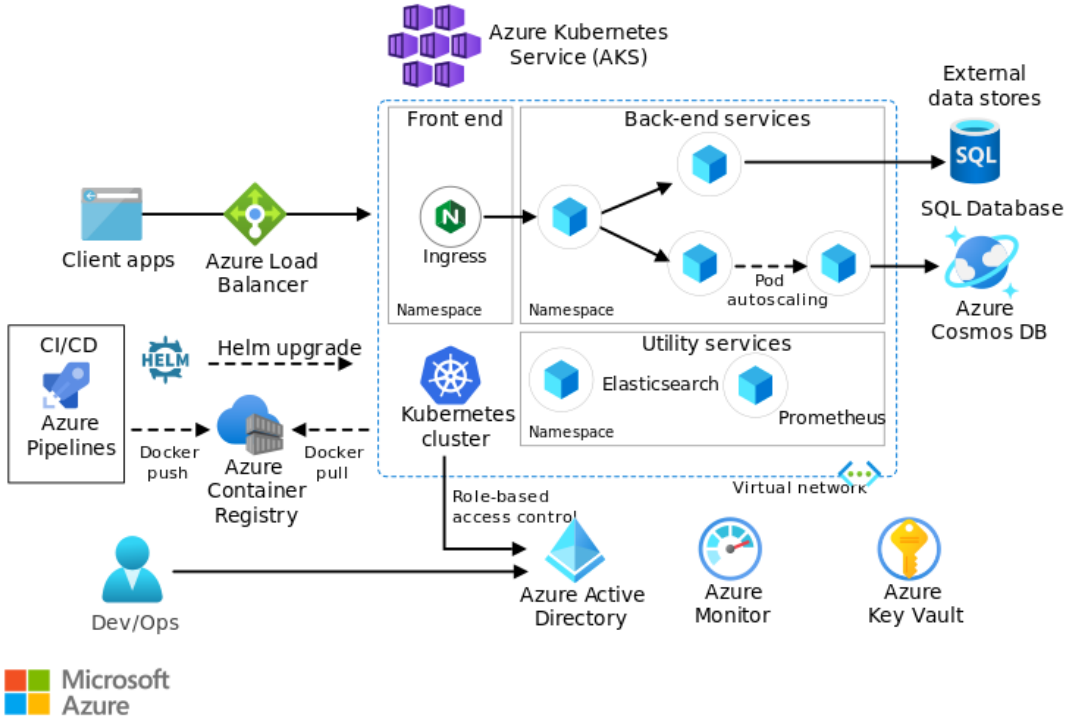
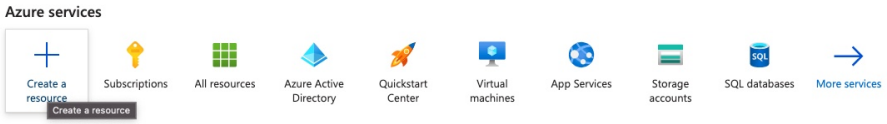


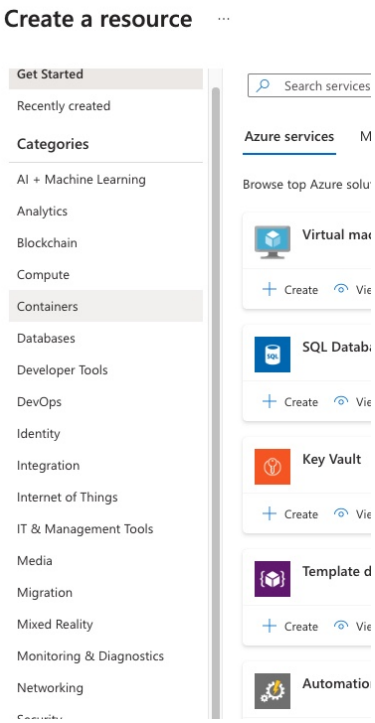

Figure 7: The landing page of Azure Kubernetes Service

To create a cluster and connect to it, we need to follow the following steps:-

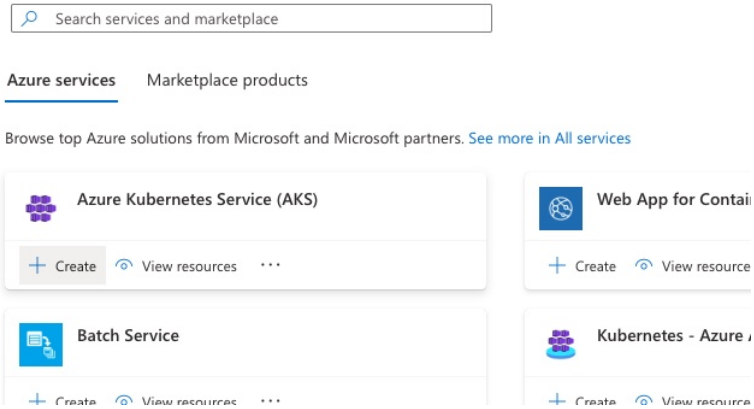
- Go to azure portal and signup.
- Go to create resource.



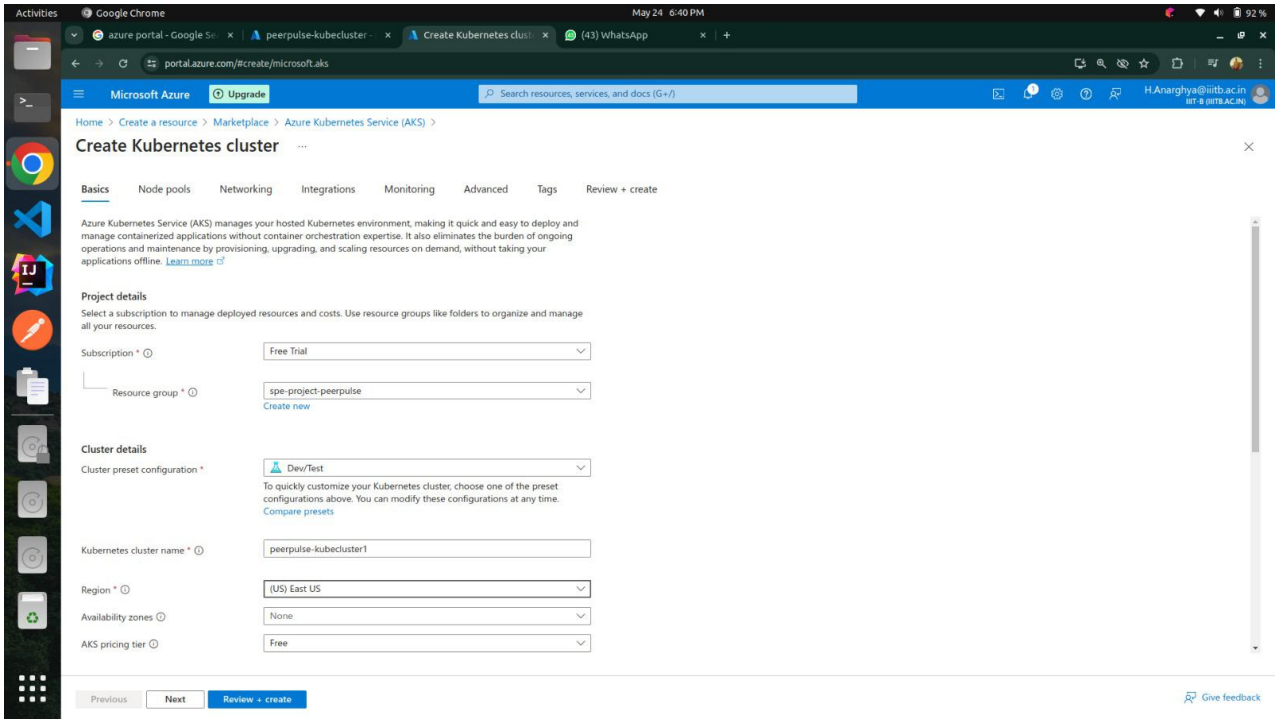

- Go to container option.



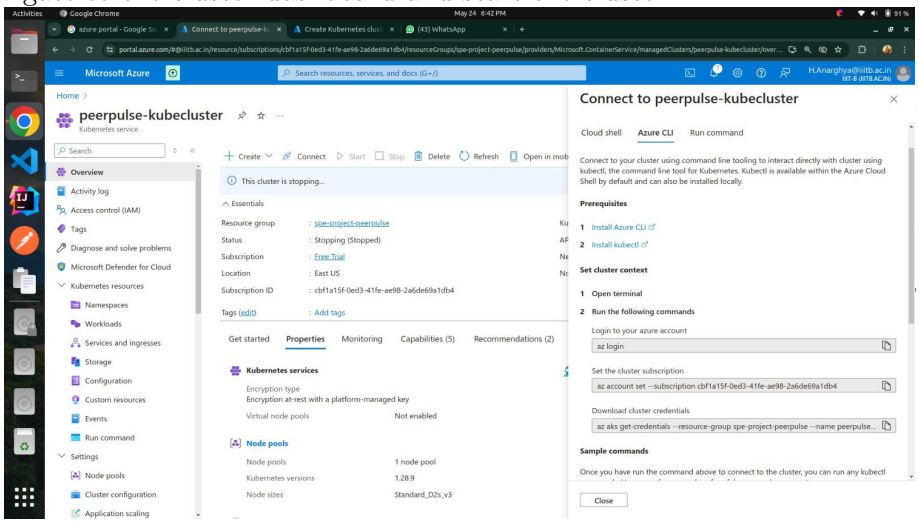

- Create a kubernetes service in it.



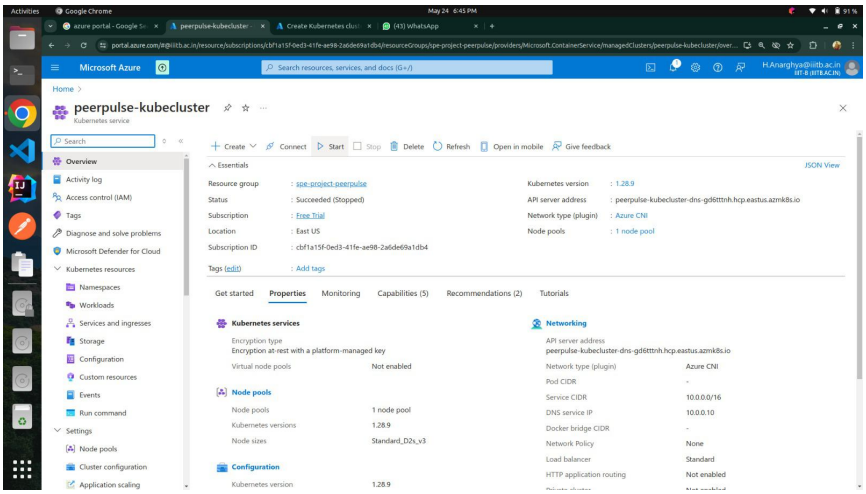

- Configure the properties of the cluster.




- Navigate to the cluster dashboard and start the cluster.




- Click on connect and write the commands needed to connect to the remote cluster. You will need azure cli installed in your system, to run these commands.



# 6 Frontend

Angular is an open-source web application framework that was initially released by Google in 2010. It is a powerful tool for building dynamic, single-page web applications and has gained popularity among developers due to its ease of use, flexibility, and robust set of features.

At its core, Angular is a framework for building client-side web applications. It provides a range of features and tools for building responsive, scalable, and user-friendly applications, including:

- **Components**: Components are the building blocks of Angular applications. They are reusable, modular pieces of code that define the structure and behavior of user interface elements. Components can be nested inside other components to create complex user interfaces.
- **Templates**: Templates are the visual representation of components. They define how the component should be rendered and include HTML, CSS, and Angular-specific syntax. Templates can be used to create dynamic user interfaces that update in real-time based on user interactions.
- **Directives**: Directives are a set of instructions that tell Angular how to manipulate the DOM. They can be used to add behavior to elements, manipulate the appearance of components, or create custom HTML tags.
- **Services**: Services are a way of encapsulating application logic and making it available throughout the application. They can be used to handle data processing, communicate with external APIs, or perform other tasks that are not directly related to the user interface.
- **Dependency Injection**: Dependency injection is a design pattern that is used to manage dependencies between different components of an application. It allows components to be loosely coupled, which makes them more modular, easier to test, and less prone to errors.

In addition to these core features, Angular also provides a range of other tools and features that make it easier to build and maintain complex web applications, including:

- **Routing**: Angular includes a powerful routing system that enables developers to create multi-page web applications with multiple views and deep linking.
- **Forms**: Angular provides a range of features for building and validating forms, including support for two-way data binding and form controls.
- **HTTP Client**: Angular includes an HTTP client that enables developers to communicate with external APIs and retrieve and manipulate data.
- **Testing**: Angular provides a range of tools and features for testing applications, including unit testing, integration testing, and end-to-end testing.

Angular is a powerful web application framework that enables developers to build dynamic, responsive, and user-friendly applications with ease. Its rich set of features, powerful tools, and flexible architecture make it a popular choice among developers, and it is likely to remain a key player in the web development landscape for years to come.

## 6.1 Installation of Angular

To install Angular on Ubuntu, we need to follow these steps:

1. **Install Node.js**: Angular requires Node.js to be installed on your system. To install Node.js, open a terminal and run the following command:

   ```
   sudo apt-get update sudo apt-get install nodejs
   ```

2. **Install npm**: npm is the package manager for Node.js and is required to install Angular. To install npm, run the following command:

   ```
   sudo apt-get install npm
   ```

3. **Install Angular CLI**: Angular CLI is a command-line interface for Angular that provides a range of tools and features for building and managing Angular applications. To install Angular CLI, run the following command:

   ```
   sudo npm install -g @angular/cli
   ```

   This command will install Angular CLI globally on your system.

4. **Verify the installation**: Once the installation is complete, you can verify that Angular CLI is installed by running the following command:

   ```
   ng version
   ```

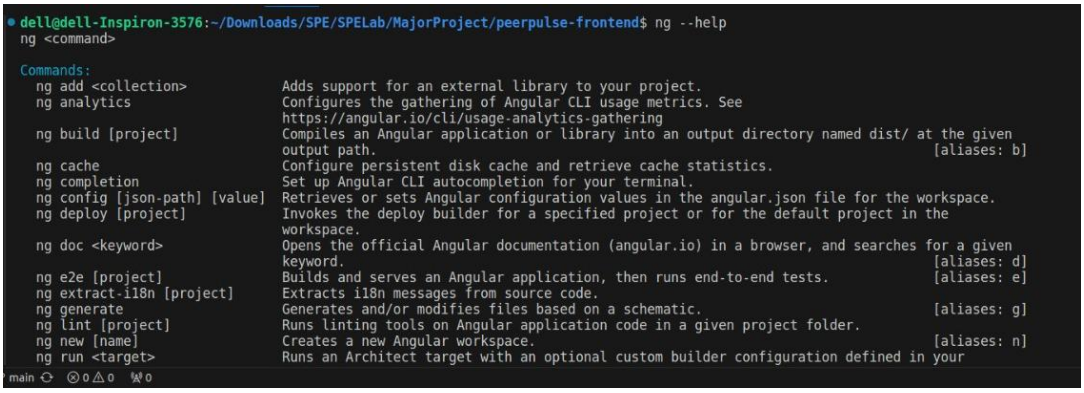



Figure 8: Completed Angular installation

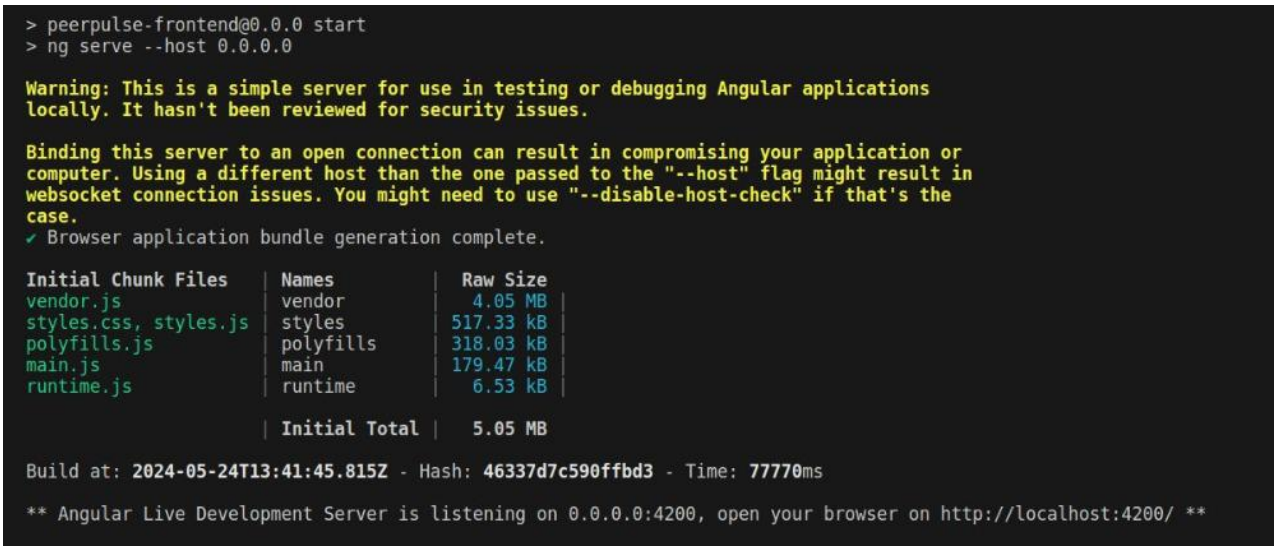



Figure 9: Running the application using `ng serve`

# 7 Source Code Management

Source code management (SCM) is the process of managing changes to the source code of a software project. SCM is an essential part of software development because it enables developers to track changes to their codebase, collaborate with others, and manage different versions of their code.

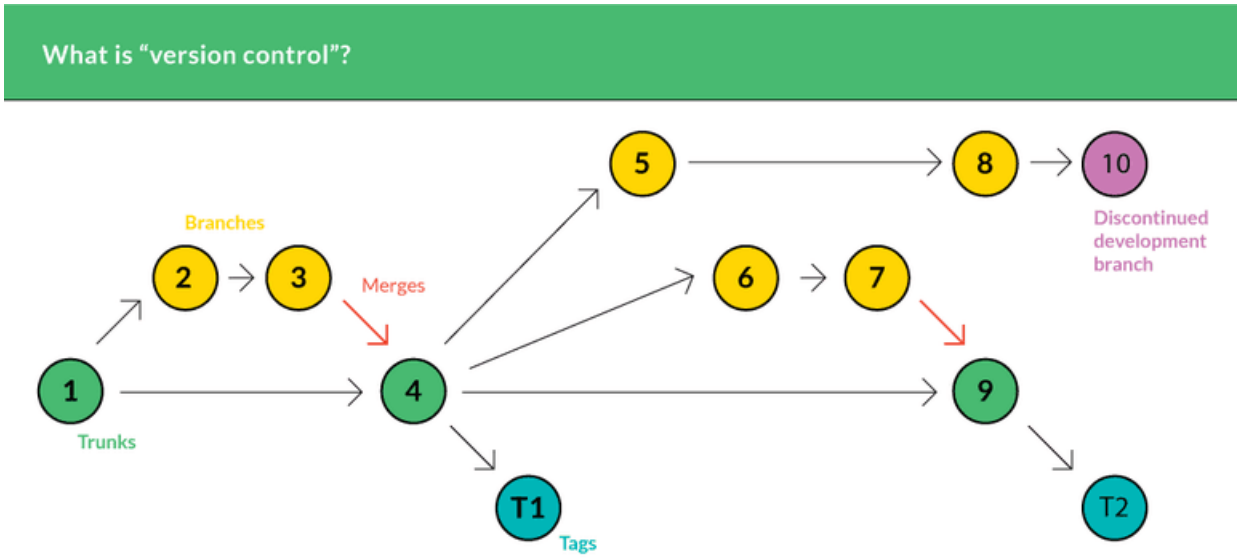



Figure 10: Representation of a version control system

SCM involves using a version control system (VCS) to manage changes to the source code. A VCS is a software system that enables developers to track changes to their codebase over time. It provides a range of features and tools for managing code changes, including:

- **Versioning**: A VCS enables developers to create different versions of their codebase, which are typically referred to as "commits". Each commit represents a snapshot of the codebase at a particular point in time.
- **Branching and Merging**: A VCS enables developers to create branches, which are separate copies of the codebase that can be modified independently. Branching allows developers to work on different features or bug fixes simultaneously without interfering with each other's work. Merging is the process of combining changes from one branch into another.
- **Collaboration**: A VCS enables developers to collaborate with others by sharing their codebase and tracking changes made by others. It provides a range of tools for managing access to the codebase, resolving conflicts, and reviewing changes made by others.
- **History**: A VCS keeps a record of all changes made to the codebase over time. This history can be used to track changes, identify bugs, and revert to earlier versions of the codebase if necessary.

There are several popular VCSs available, including Git, Subversion, and Mercurial. Git is one of the most widely used VCSs and is the de facto standard for modern software development.

SCM is essential for software development because it enables developers to work collaboratively on large codebases without creating conflicts or introducing errors. It also provides a range of tools for managing changes, reviewing code, and tracking progress, which can improve the quality and reliability of software projects.

In addition to its benefits for software development, SCM has become increasingly important for DevOps and cloud computing. In DevOps, SCM is used to manage changes to infrastructure code and automate deployment processes. In cloud computing, SCM is used to manage changes to cloud resources and ensure that changes are tracked and audited.

Source code management is an essential part of software development that provides a range of benefits for developers, including improved collaboration, better code quality, and increased productivity. By leveraging the tools and features provided by version control systems, developers can build better software more efficiently and with fewer errors.

## 7.1 Git

Git is a distributed version control system (VCS) that is widely used for software development. It was created by Linus Torvalds in 2005 and has since become one of the most popular VCSs in use today.

Git is designed to manage changes to large codebases and enable collaboration among developers. It provides a range of features and tools for managing code changes, including:

- **Versioning**: Git enables developers to create different versions of their codebase, which are typically referred to as "commits". Each commit represents a snapshot of the codebase at a particular point in time.
- **Branching and Merging**: Git enables developers to create branches, which are separate copies of the codebase that can be modified independently. Branching allows developers to work on different features or bug fixes simultaneously without interfering with each other's work. Merging is the process of combining changes from one branch into another.
- **Collaboration**: Git enables developers to collaborate with others by sharing their codebase and tracking changes made by others. It provides a range of tools for managing access to the codebase, resolving conflicts, and reviewing changes made by others.
- **History**: Git keeps a record of all changes made to the codebase over time. This history can be used to track changes, identify bugs, and revert to earlier versions of the codebase if necessary.

Git is a distributed VCS, which means that each developer has a complete copy of the codebase on their local machine. This makes it easy for developers to work offline and enables them to make changes to the codebase without having to be connected to a central server.

Another key feature of Git is its ability to handle large codebases. Git is optimized for speed and can handle codebases with hundreds of thousands of files and millions of lines of code. This makes it an ideal VCS for large software projects.

Git is also highly flexible and can be customized to meet the needs of different software development workflows. It supports a range of branching and merging strategies, and can be integrated with other tools and services, such as continuous integration and deployment (CI/CD) systems.

Git is a powerful and flexible VCS that provides a range of features and tools for managing changes to large codebases and enabling collaboration among developers. Its popularity has made it a de facto standard for modern software development, and it is likely to remain a key tool for developers for years to come.

### 7.1.1 Basic Commands

Some popular Git commands are:

- **git init**: This command initializes a new Git repository. You can use it to create a new repository from scratch or to initialize an existing project as a Git repository.

  ```
  git init my-project
  ```

- **git add**: This command adds files to the staging area, which prepares them to be committed to the repository. Here's an example:

  ```
  git add file1.txt file2.txt
  ```

  This command adds "file1.txt" and "file2.txt" to the staging area.

- **git commit**: This command creates a new commit with the changes in the staging area. Here's an example:

  ```
  git commit -m "Added new feature"
  ```

  This command creates a new commit with the changes in the staging area and adds a commit message "Added new feature".

- **git push**: This command pushes the local changes to a remote repository. Here's an example:

  ```
  git push origin main
  ```

  This command pushes the local changes to the "main" branch of the "origin" remote repository.

- **git pull**: This command fetches changes from a remote repository and merges them into the local repository. Here's an example:

  ```
  git pull origin main
  ```

  This command fetches changes from the "main" branch of the "origin" remote repository and merges them into the local repository.

- **git branch**: This command lists all the branches in the repository or creates a new branch. Here's an example:

  ```
  git branch feature-branch
  ```

  This command creates a new branch called "feature-branch".

- **git checkout**: This command switches between branches or restores files from a previous commit. Here's an example:

  ```
  git checkout feature-branch
  ```

  This command switches the current branch to "feature-branch".

## 7.2 GitHub

GitHub is a web-based hosting service for version control using Git. It was created in 2008 and is now owned by Microsoft. GitHub provides a platform for developers to collaborate on software projects, share code, and contribute to open source projects.

GitHub provides a range of features and tools for managing software projects, including:

- **Version Control**: GitHub provides a powerful version control system using Git. This enables developers to manage changes to their codebase, track bugs, and revert to earlier versions of the codebase if necessary.

- **Collaboration**: GitHub provides a range of tools for collaboration, including pull requests, issues, and project boards. Pull requests enable developers to propose changes to the codebase and get feedback from other developers before merging the changes into the main codebase. Issues can be used to track bugs, feature requests, and other tasks. Project boards provide a way to track the progress of a project and manage tasks.
- **Code Reviews**: GitHub provides a code review system that enables developers to review each other's code before merging it into the main codebase. Code reviews can improve code quality, identify bugs, and ensure that the codebase meets the standards of the project.
- **Continuous Integration and Deployment**: GitHub can be integrated with a range of continuous integration and deployment (CI/CD) systems, such as Jenkins and Travis CI. This enables developers to automate the process of building, testing, and deploying software.
- **Open Source**: GitHub is a popular platform for open source projects. It provides a range of tools and features for managing open source projects, including licensing, code of conduct, and contribution guidelines.

GitHub also provides a range of community features, such as profiles, activity feeds, and social networking. This enables developers to connect with each other, share knowledge, and build relationships.

GitHub is a powerful platform for managing software projects and collaborating with other developers. Its powerful version control system, collaboration tools, and community features make it an essential tool for modern software development. It is widely used by developers around the world, and has become a central hub for the open source community.

## 7.3 Creation of GitHub repository

The repository can be created on GitHub, as shown below.

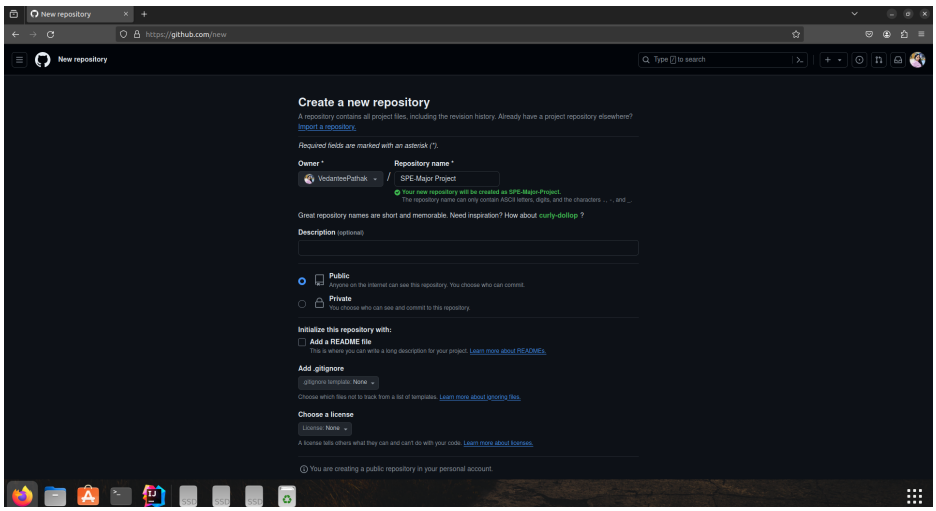



Figure 11: Creation of a repository on GitHub

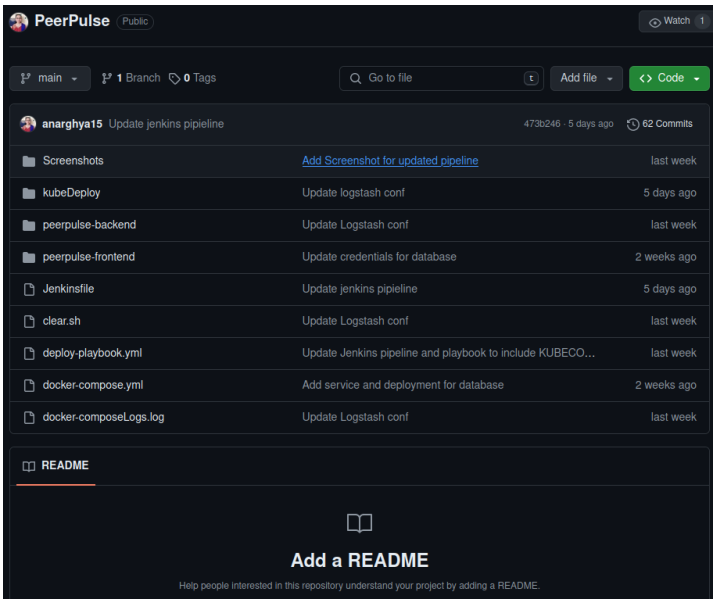



Figure 12: View of the repository on GitHub

The GitHub URL can be added to our local git system using `git remote add origin <repo URL>`. A commit can be made using `git commit -m "a meaningful commit message"` and local filesystem changes can be sent to GitHub using `git push origin main`.


Figure 13: A view of the commits on GitHub

The GitHub repository can be found at https://github.com/anarghya15/PeerPulse.git, **not publicly accessible at the moment**.

## 8 Continuous Integration

Continuous integration (CI) is a software development practice that involves automatically building, testing, and integrating code changes into a shared repository on a regular basis. The goal of CI is to catch code errors and conflicts early in the development process, reducing the time and cost required to fix them.

In a typical CI workflow, developers regularly commit their changes to a shared repository, which triggers a series of automated tests and checks. If any issues are detected, developers are notified immediately so that they can fix the issue before it causes further problems.

CI is typically implemented using a continuous integration server or platform, such as Jenkins,

Travis CI, or CircleCI. These platforms provide a range of features and tools for building, testing, and deploying software, including:

- **Build Automation**: CI platforms automatically build the software whenever new changes are committed to the repository. This ensures that the software is always up-to-date and that all changes are properly integrated.
- **Automated Testing**: CI platforms run a suite of automated tests to ensure that the software is functioning correctly. This includes unit tests, integration tests, and end-to-end tests.
- **Code Quality Checks**: CI platforms check the code for code quality issues, such as coding standards violations, security vulnerabilities, and performance problems.
- **Notification and Reporting**: CI platforms provide notifications and reports to developers when issues are detected. This enables developers to quickly identify and fix issues before they become more serious.

By implementing CI, software development teams can benefit from a range of advantages, including:

- **Reduced Time to Market**: CI enables developers to catch issues early in the development process, reducing the time and cost required to fix them. This enables software to be delivered to customers more quickly.
- **Improved Code Quality**: CI ensures that code is regularly tested and integrated, improving code quality and reducing the risk of bugs and issues.
- **Faster Feedback**: CI provides developers with fast feedback on their changes, enabling them to quickly identify and fix issues before they become more serious.
- **Increased Collaboration**: CI encourages collaboration among developers by ensuring that all changes are integrated into a shared repository on a regular basis.

Continuous integration is an essential practice for modern software development. By automating the build, testing, and integration process, CI enables teams to catch issues early in the development process, improve code quality, and deliver software more quickly and efficiently.

### 8.0.1 Jenkins

Jenkins is an open-source automation server that is used for continuous integration and continuous delivery (CI/CD) of software. It was created in 2004 and has since become one of the most widely used automation servers in the world.

Jenkins is designed to automate the software development process, from building and testing code to deploying it to production. It provides a range of features and tools for managing software projects, including:

- **Continuous Integration**: Jenkins provides built-in support for continuous integration, enabling developers to automatically build and test their code whenever changes are committed to the repository. This ensures that the software is always up-to-date and that all changes are properly integrated.
- **Continuous Delivery**: Jenkins provides support for continuous delivery, enabling developers to automatically deploy their code to production environments whenever it passes automated tests and meets predefined criteria.

- **Extensibility**: Jenkins is highly extensible and can be customized to meet the needs of different software development workflows. It provides a range of plugins and integrations with other tools and services, such as Git, GitHub, Docker, and Kubernetes.
- **Scalability**: Jenkins is designed to be highly scalable, enabling it to handle large and complex software projects. It can be run on a single server or distributed across multiple servers to handle high-volume workloads.
- **Security**: Jenkins provides a range of security features to protect software projects from unauthorized access and attacks. This includes user authentication and authorization, role-based access control, and secure communication protocols.

**Jenkins**

Jenkins is highly customizable and can be configured to meet the needs of different software development workflows. It provides a range of plugins and integrations with other tools and services, such as version control systems, bug trackers, and deployment tools. This makes it an ideal automation server for modern software development.

Jenkins also provides a range of community features, such as forums, user groups, and conferences. This enables developers to connect with each other, share knowledge, and build relationships.

Jenkins is a powerful and flexible automation server that provides a range of features and tools for managing software projects. Its support for continuous integration and continuous delivery, scalability, and extensibility make it an essential tool for modern software development. Its popularity has made it a de facto standard for CI/CD and automation in the software industry.

Jenkins is a powerful automation server that provides a range of features and tools for managing software projects. While it offers many benefits, there are also some disadvantages to consider when using Jenkins. In this answer, we will discuss the advantages and disadvantages of using Jenkins.

Some advantages of using Jenkins:

- **Continuous Integration**: Jenkins provides built-in support for continuous integration, enabling developers to automatically build and test their code whenever changes are committed to the repository. This ensures that the software is always up-to-date and that all changes are properly integrated.
- **Continuous Delivery**: Jenkins provides support for continuous delivery, enabling developers to automatically deploy their code to production environments whenever it passes automated tests and meets predefined criteria.
- **Extensibility**: Jenkins is highly extensible and can be customized to meet the needs of different software development workflows. It provides a range of plugins and integrations with other tools and services, such as Git, GitHub, Docker, and Kubernetes.

- **Scalability**: Jenkins is designed to be highly scalable, enabling it to handle large and complex software projects. It can be run on a single server or distributed across multiple servers to handle high-volume workloads.
- **Security**: Jenkins provides a range of security features to protect software projects from unauthorized access and attacks. This includes user authentication and authorization, role-based access control, and secure communication protocols.
- **Community Support**: Jenkins has a large and active community of developers who contribute to its development, provide support, and share knowledge. This ensures that Jenkins is constantly evolving and improving.
- **Open Source**: Jenkins is an open-source project, which means that it is free to use and can be modified and distributed by anyone. This makes it an accessible and affordable option for small and large software development teams alike.

On the other hans, some disadvantages of using Jenkins:

- **Complexity**: Jenkins can be complex to set up and configure, particularly for new users. It requires a certain level of technical knowledge and expertise to get started.
- **Maintenance**: Jenkins requires regular maintenance and updates to ensure that it is running smoothly and securely. This can be time-consuming and may require dedicated resources.
- **Plugin Management**: Jenkins relies heavily on plugins to provide its functionality. Managing plugins and ensuring that they are up-to-date and compatible with each other can be challenging.
- **Resource Intensive**: Jenkins can be resource-intensive, particularly when running large and complex software projects. This can lead to performance issues and may require dedicated hardware resources.
- **Limited Support**: While Jenkins has a large and active community, support is primarily provided by the community rather than a dedicated support team. This may limit the availability of support and resources for certain issues.

In conclusion, Jenkins offers many benefits for software development teams, including continuous integration and delivery, extensibility, scalability, security, and community support. However, it also has some disadvantages, such as complexity, maintenance requirements, plugin management, resource intensity, and limited support. When considering whether to use Jenkins, it is important to weigh these advantages and disadvantages and consider whether Jenkins is the right automation server for your specific needs and requirements.

## 8.1 Installation of Jenkins

To install Jenkins on Ubuntu, follow these steps:

1. **Update Ubuntu**

   Before installing Jenkins, it's important to update Ubuntu to ensure that all packages are up-to-date. Use the following command to update Ubuntu:

   ```
   sudo apt-get update
   ```

2. **Install Java**

   Jenkins requires Java to run. Use the following command to install Java on Ubuntu:

   ```
   sudo apt-get install default-jdk
   ```

3. **Add Jenkins Repository Key**

   Next, add the Jenkins repository key to your system using the following commands:

   ```
   wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
   ```

   This command downloads the Jenkins repository key and adds it to the system's trusted keys list.

4. **Add Jenkins Repository**

   Once the repository key is added, add the Jenkins repository to your system using the following command:

   ```
   sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/
   > /etc/apt/sources.list.d/jenkins.list'
   ```

   This command adds the Jenkins repository to the system's list of package sources.

5. **Install Jenkins**

   Now that the Jenkins repository has been added, use the following commands to install Jenkins:

   ```
   sudo apt-get update sudo apt-get install jenkins
   ```

   This will install Jenkins on your Ubuntu system.

6. **Start Jenkins**

   After the installation is complete, use the following command to start the Jenkins service:

   ```
   sudo systemctl start jenkins
   ```

   This command starts the Jenkins service on your Ubuntu system.

7. **Enable Jenkins**

   Once the Jenkins service is started, use the following command to enable it to start automatically at boot:

   ```
   sudo systemctl enable jenkins
   ```

   This command configures the Jenkins service to start automatically at boot time.

8. **Access Jenkins**

   You can now access Jenkins by opening a web browser and navigating to `http://localhost:8080`. If you are accessing Jenkins from a remote machine, replace "localhost" with the IP address or hostname of the Ubuntu system.

9. **Unlock Jenkins**

   When you access Jenkins for the first time, you will be prompted to enter an initial admin password. Use the following command to retrieve the password:

   ```
   sudo cat /var/lib/jenkins/secrets/initialAdminPassword
   ```

   This command retrieves the initial admin password from the Jenkins installation directory. Copy and paste the password into the Jenkins web interface to unlock Jenkins.

Jenkins has been successfully installed and configured on a Ubuntu system.

## 8.2 ngrok and Webhooks

ngrok is a tool that allows developers to expose a web server running on their local machine to the internet. It creates a secure tunnel between the local machine and a public URL that can be accessed from anywhere, making it possible to test and share applications without the need for a public-facing server.

When ngrok is running, it provides a temporary public URL that can be used to access the local server. This URL can be shared with others, allowing them to view and interact with the application being developed. The ngrok tunnel is encrypted using HTTPS, ensuring that traffic between the local machine and the public URL is secure. Developers can use ngrok to receive webhooks from external services during development and testing.

Run the following command to start Ngrok: `./ngrok http <port-number>`. Replace `<port-number>` with the port number of the local web server we want to expose. Ngrok will generate a public URL that we can use to access our local web server from anywhere. We can find the URL in the Ngrok terminal window under the "Forwarding" section. Copy the URL and paste it into our web browser to test it. The local web server is now accessible from the internet using the Ngrok public URL. Keep in mind that the Ngrok session will end if we close the terminal window or stop the Ngrok process.

GitHub webhooks can be used in Jenkins to trigger builds or perform other actions in response to events in a GitHub repository, such as a new pull request being opened or a code push to a specific branch.

To set up GitHub webhooks in Jenkins, we need to follow these steps:

- Install the GitHub plugin in Jenkins, which allows Jenkins to interact with GitHub repositories and respond to webhook events.
- Create a new Jenkins job or configure an existing one to respond to a webhook event. Select the "GitHub hook trigger for GITScm polling" option in the Build Triggers section of the job configuration.
- In the GitHub repository settings, add a new webhook and specify the URL of the Jenkins server and the payload URL for the specific webhook event. You can choose to trigger the webhook event for all pushes, pull requests, or specific branches or tags.
- Save the webhook configuration and test the integration by triggering a webhook event in the GitHub repository. Jenkins should receive the event and trigger a build or perform the specified action.

GitHub webhooks in Jenkins can be used to automate various aspects of the development workflow, such as building and testing code, deploying applications, and notifying team members of new events. By using webhooks, we can streamline the development process and reduce manual effort.

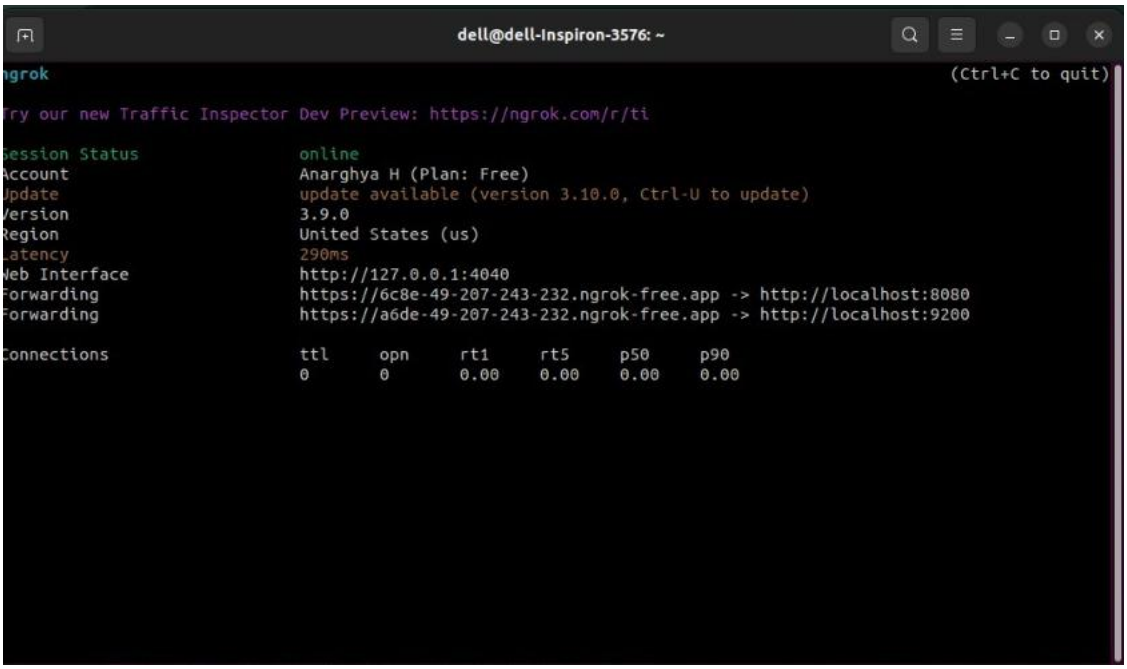



Figure 14: Running ngrok to obtain the forwarding URL

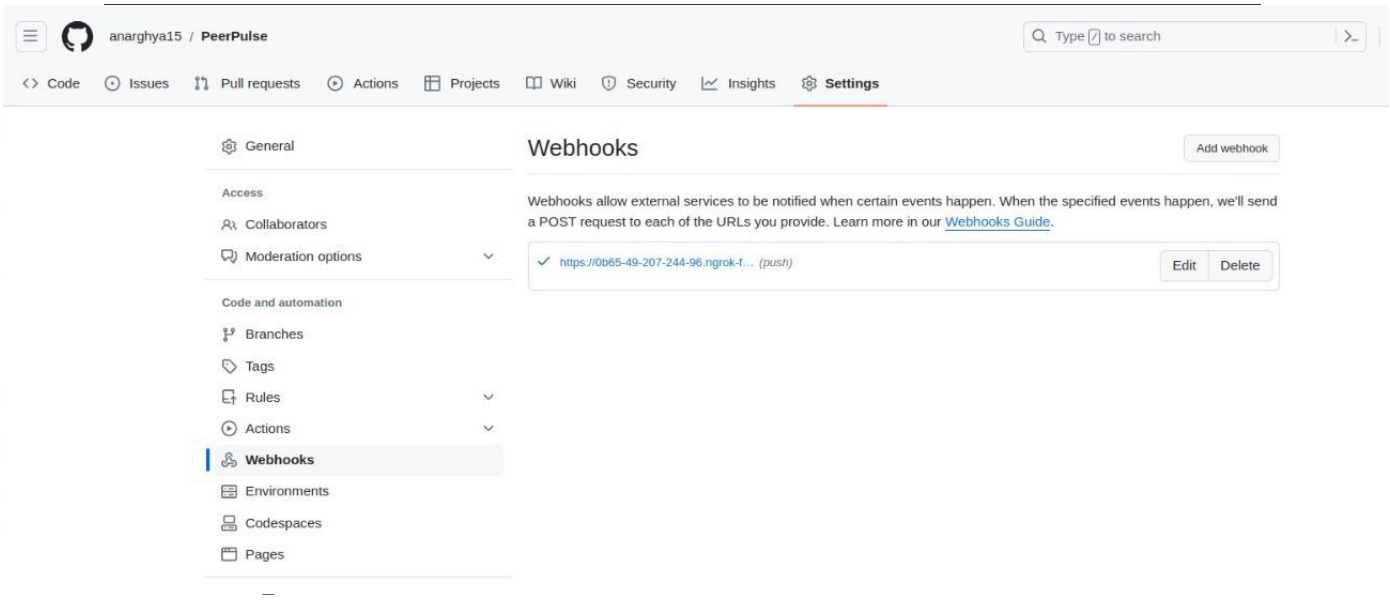



Figure 15: Choosing the webhooks options on the GitHub repository

In Jenkins, there are several types of build triggers available to automatically start a build process:

- **Polling SCM**: This trigger regularly checks the source code repository for changes and starts a build if new changes are detected.
- **Schedule**: This trigger starts a build at a specified time or on a recurring schedule.
- **Build after other projects are built**: This trigger starts a build after one or more specified projects are built successfully.
- **Trigger builds remotely**: This trigger allows remote programs or scripts to start a build through a URL or API call.
- **Build periodically**: This trigger starts a build at regular intervals, such as every hour or every day.

- **GitHub hook trigger for GITScm polling**: This trigger allows Jenkins to listen for incoming webhooks from GitHub and start a build when changes are pushed to the repository.
- **Pipeline trigger**: This trigger allows one pipeline to trigger another pipeline.

These triggers can be configured in the Jenkins project settings, and multiple triggers can be used in combination to start a build based on different conditions.

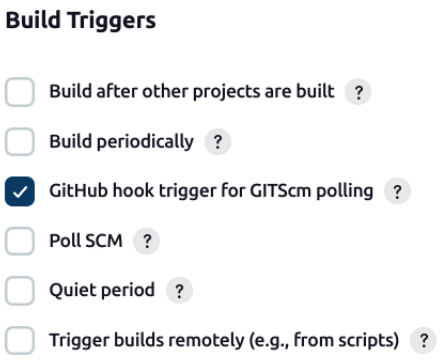



Figure 16: Choosing the GitSCM polling mechanism in Jenkins

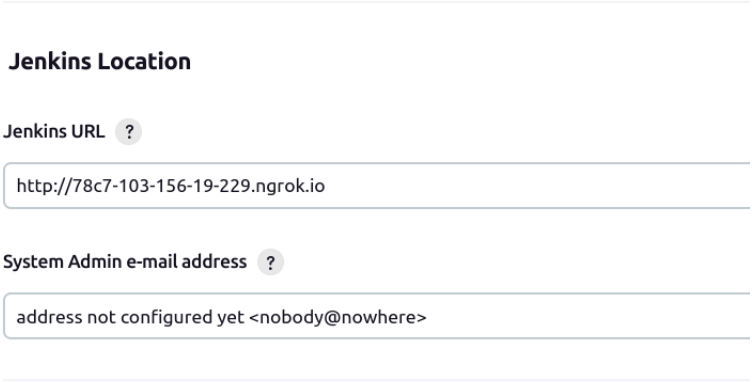



Figure 17: Adding the ngrok forwarding URL to Jenkins

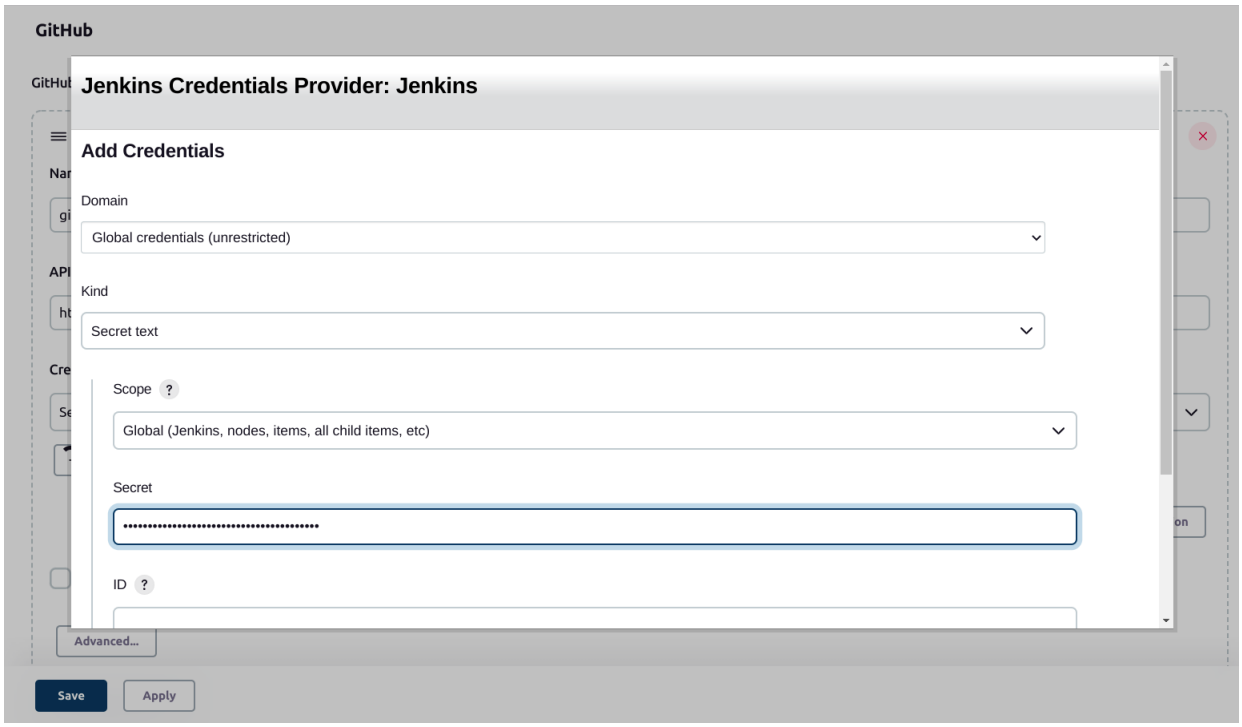



Figure 18: Adding the secret text generated from GitHub to the Jenkins Credentials provider

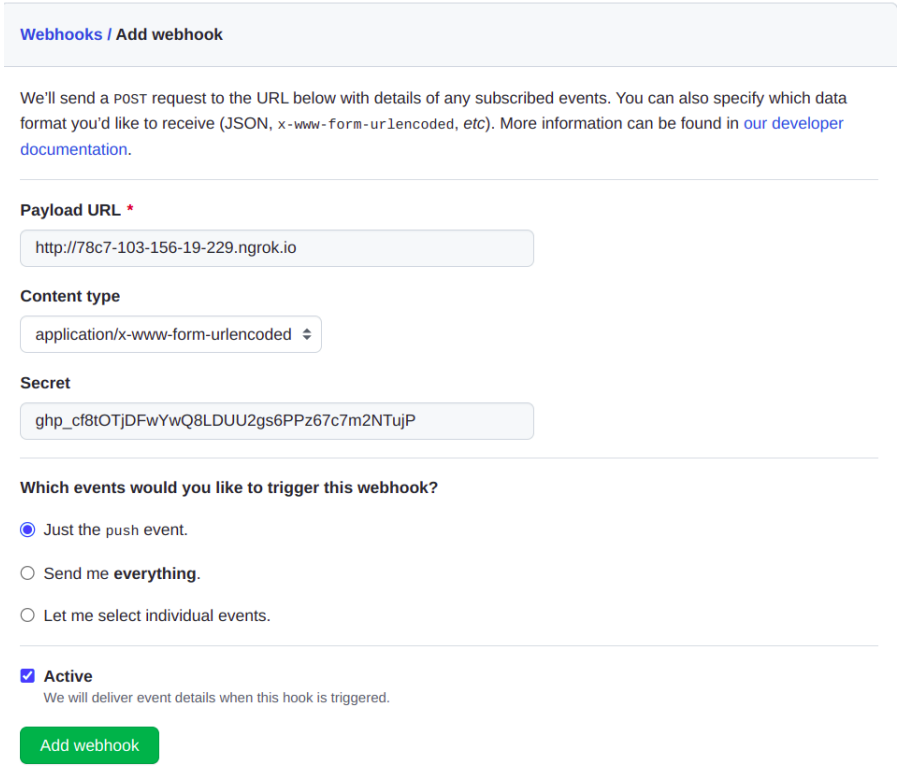



Figure 19: Adding the ngrok forwarding URL to the GitHub repository

# 9 Application Testing

Testing is an important part of the software development process in Spring Boot, and can help to ensure that the application works as intended and is free from errors and bugs.

Unit testing is the process of testing individual units of code, such as methods or functions, to ensure they work as intended. In Spring Boot, unit testing can be done using JUnit, a popular testing framework. JUnit tests can be run automatically as part of the build process, and can test individual components of the application in isolation.

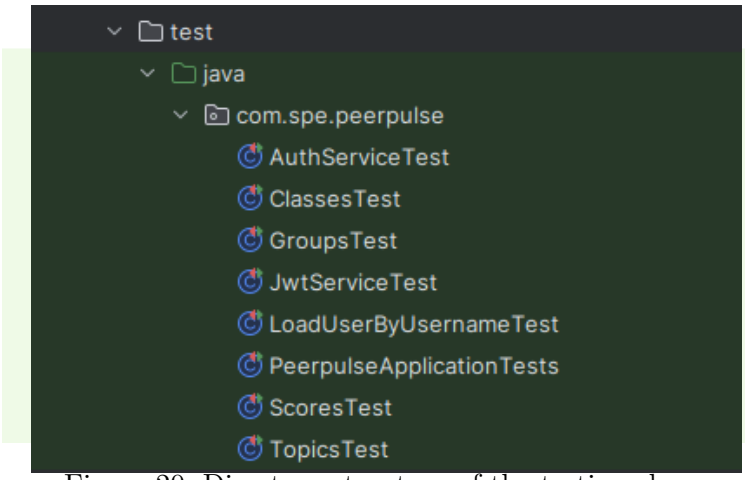



Figure 20: Directory structure of the testing classes

Each of the services in the the backend is tested appropriately using JUnit5 unit testing cases.

```
@Test
void testFindAllTopics()
{
    Long classId = 1L;

    // mock the repository method to return the test data
    when(topicRepository.findByClassObj_ClassId(classId)).thenReturn(topics);

    // call the service method
    List<Topic> result = teacherService.findAllTopics(classId);

    // assert that the result matches the test data
    assertEquals(topics, result);

    // verify that the repository method was called with the correct argument
    verify(topicRepository).findByClassObj_ClassId(classId);
}
```

Figure 21: A test case for the function `findAllTopics()`

This is a test class written in Java for a Spring Boot application. It tests the function-

ality of the `findAllTopics` method in the `TeacherService` class. The test uses the Mockito library to mock the `TopicRepository` and inject it into the `TeacherService` class. The `setUp` method is used to create some test data in the form of a list of Topic objects. The `testFindAllTopics` method tests the `findAllTopics` method of the `TeacherService` class by mocking the `findByClassObj_ClassId` method of the `TopicRepository` class to return the test data. The test then calls the `findAllTopics` method and asserts that the returned result matches the test data. Finally, it verifies that the `findByClassObj_ClassId` method was called with the correct argument.

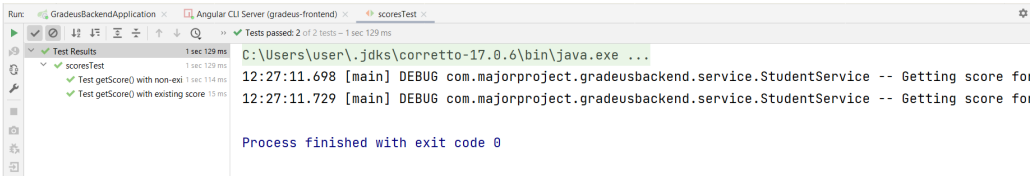



Figure 22: A run of the `scoresTest` class

## 10 Containerization

Containerization is a method of packaging and deploying software applications in a lightweight and portable manner. It involves creating a container that encapsulates an application and all its dependencies, allowing it to run consistently across different environments and platforms.

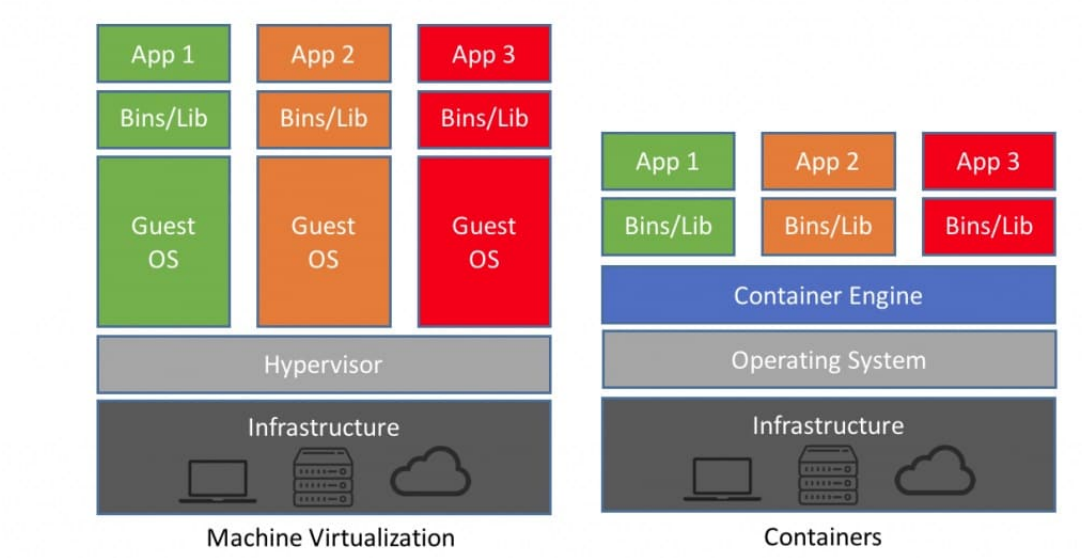



Figure 23: Virtualization vs Containerization

Containers are isolated from the underlying operating system and other containers, enabling multiple applications to run simultaneously without interfering with each other. Each container has its own file system, network interfaces, and resource allocation, making it a self-contained unit that can be easily moved or replicated.

Containerization has become increasingly popular in recent years, particularly in the context of modern software development and deployment. Some of the key benefits of containerization include:

- **Consistency**: Containers provide a consistent runtime environment across different platforms and environments. This reduces the risk of compatibility issues and makes it easier to deploy and manage software applications.

- **Portability**: Containers can be easily moved between different environments, such as from development to production or from one cloud provider to another. This makes it easier to scale and distribute software applications as needed.
- **Efficiency**: Containers are lightweight and require minimal system resources compared to traditional virtual machines. This enables more efficient use of hardware resources and reduces the cost of running and maintaining software applications.
- **Security**: Containers provide a level of isolation and security that helps to protect against malicious attacks and vulnerabilities. Each container runs in its own sandboxed environment, minimizing the impact of any security breaches.

Some popular containerization technologies include Docker, Kubernetes, and Red Hat OpenShift. These platforms provide a range of features and tools for building, deploying, and managing containerized applications.

Containerization is a powerful and flexible approach to software deployment that offers a range of benefits for modern software development. By encapsulating applications and their dependencies in self-contained units, containers enable more efficient and consistent deployment of software applications across different environments and platforms.

## 10.1 Docker

Docker is an open-source platform that allows developers to build, ship, and run applications in containers. Containers are lightweight, portable, and self-contained units that encapsulate an application and all its dependencies, enabling it to run consistently across different environments and platforms.

Docker was first introduced in 2013 and has since become one of the most widely used containerization platforms. It provides a range of features and tools for containerization, including:

- **Docker Engine**: This is the core component of Docker that enables users to build, run, and manage containers. It provides a runtime environment for containers, as well as tools for managing container images, networks, and storage.
- **Docker Hub**: This is a central repository for Docker images, providing access to a wide range of pre-built images that can be used as the basis for building containers.
- **Docker Compose**: This is a tool for defining and managing multi-container applications. It enables developers to define a set of services and their dependencies in a single file, simplifying the deployment and management of complex applications.
- **Docker Swarm**: This is a tool for managing clusters of Docker hosts. It enables developers to deploy and manage a large number of containers across multiple hosts, providing high availability and scalability for containerized applications.

Docker has become increasingly popular in recent years, particularly in the context of modern software development and deployment. Some of the key benefits of using Docker include:

- **Consistency**: Docker enables developers to create a consistent runtime environment for their applications, ensuring that they run the same way across different environments and platforms.
- **Portability**: Docker containers can be easily moved between different environments, such as from development to production or from one cloud provider to another. This makes it easier to scale and distribute software applications as needed.

- **Efficiency**: Docker containers are lightweight and require minimal system resources compared to traditional virtual machines. This enables more efficient use of hardware resources and reduces the cost of running and maintaining software applications.
- **Security**: Docker provides a level of isolation and security that helps to protect against malicious attacks and vulnerabilities. Each container runs in its own sandboxed environment, minimizing the impact of any security breaches.

Docker is a powerful and flexible platform for containerization that offers a range of benefits for modern software development. By encapsulating applications and their dependencies in self-contained units, Docker enables more efficient and consistent deployment of software applications across different environments and platforms.

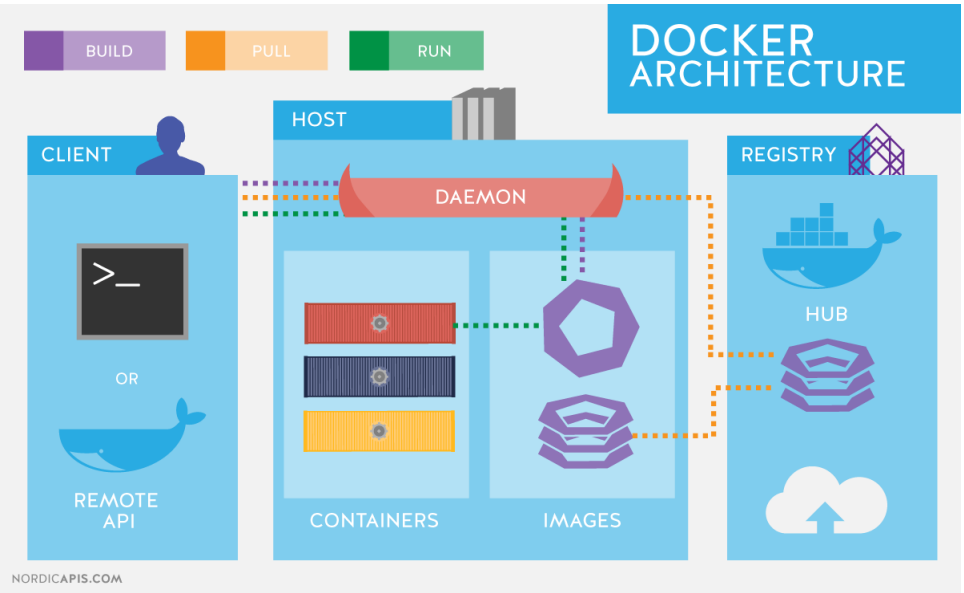



Figure 24: Docker Architecture

Docker is a popular containerization platform that provides a range of benefits for modern software development and deployment. While it offers many advantages, there are also some disadvantages to consider when using Docker. Some advantages of using Docker:

- **Consistency**: Docker provides a consistent runtime environment for applications, ensuring that they run the same way across different environments and platforms. This reduces the risk of compatibility issues and makes it easier to deploy and manage software applications.
- **Portability**: Docker containers can be easily moved between different environments, such as from development to production or from one cloud provider to another. This makes it easier to scale and distribute software applications as needed.
- **Efficiency**: Docker containers are lightweight and require minimal system resources compared to traditional virtual machines. This enables more efficient use of hardware resources and reduces the cost of running and maintaining software applications.
- **Security**: Docker provides a level of isolation and security that helps to protect against malicious attacks and vulnerabilities. Each container runs in its own sandboxed environment, minimizing the impact of any security breaches.
- **Scalability**: Docker enables developers to easily scale their applications up or down as needed. Containers can be easily replicated and deployed across multiple hosts, providing high availability and scalability for containerized applications.

- **Version Control**: Docker enables developers to easily version their applications and dependencies, making it easier to roll back to a previous version if needed.
- **Community Support**: Docker has a large and active community of developers who contribute to its development, provide support, and share knowledge. This ensures that Docker is constantly evolving and improving.
- **Open Source**: Docker is an open-source project, which means that it is free to use and can be modified and distributed by anyone. This makes it an accessible and affordable option for small and large software development teams alike.

Some disadvantages to consider while using Docker:

- **Complexity**: Docker can be complex to set up and configure, particularly for new users. It requires a certain level of technical knowledge and expertise to get started.
- **Maintenance**: Docker requires regular maintenance and updates to ensure that it is running smoothly and securely. This can be time-consuming and may require dedicated resources.
- **Plugin Management**: Docker relies heavily on plugins to provide its functionality. Managing plugins and ensuring that they are up-to-date and compatible with each other can be challenging.
- **Resource Intensive**: Docker can be resource-intensive, particularly when running large and complex software projects. This can lead to performance issues and may require dedicated hardware resources.
- **Learning Curve**: Docker requires developers to learn a new set of concepts and tools, which can be challenging for those who are unfamiliar with containerization.
- **Limited Support**: While Docker has a large and active community, support is primarily provided by the community rather than a dedicated support team. This may limit the availability of support and resources for certain issues.

Docker offers many benefits for software development teams, including consistency, portability, efficiency, security, scalability, version control, community support, and open-source availability. However, it also has some disadvantages, such as complexity, maintenance requirements, plugin management, resource intensity, learning curve, and limited support. When considering whether to use Docker, it is important to weigh these advantages and disadvantages and consider whether Docker is the right containerization platform for your specific needs and requirements.

### 10.1.1 Installation

To install Docker on Ubuntu, follow these steps:

1. **Update Ubuntu**

   Before installing Docker, it is important to update Ubuntu to ensure that all packages are up-to-date. Use the following command to update Ubuntu:

   ```
   sudo apt-get update
   ```

2. **Install Dependencies**

   Docker requires a few dependencies to be installed on Ubuntu. Use the following command to install these dependencies:

```
sudo apt-get install apt-transport-https ca-certificates
curl gnupg-agent software-properties-common
```

This command installs the necessary packages for Docker to work properly.

3. **Add Docker Repository Key**

   Next, add the Docker repository key to your system using the following commands:

   ```
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
   ```

   This command downloads the Docker repository key and adds it to the system's trusted keys list.

4. **Add Docker Repository**

   Once the repository key is added, add the Docker repository to your system using the following command:

   ```
   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
   \$(lsb_release -cs) stable"
   ```

   This command adds the Docker repository to the system's list of package sources.

5. **Install Docker**

   Now that the Docker repository has been added, use the following commands to install Docker:

   ```
   sudo apt-get update sudo apt-get install docker-ce docker-ce-cli containerd.io
   ```

   This will install Docker on your Ubuntu system.

6. **Start Docker**

   After the installation is complete, use the following command to start the Docker service:

   `sudo systemctl start docker` This command starts the Docker service on your Ubuntu system.

7. **Enable Docker**

   Once the Docker service is started, use the following command to enable it to start automatically at boot:

   ```
   sudo systemctl enable docker
   ```

   This command configures the Docker service to start automatically at boot time.

8. **Verify Docker Installation**

   You can now verify that Docker is installed and running on your Ubuntu system by running the following command:

   ```
   sudo docker run hello-world
   ```

   This command will download and run a simple "Hello World" Docker image, confirming that Docker is installed and working properly.

We have successfully installed and configured Docker on your Ubuntu system.

## 10.2 Docker Hub

Docker Hub is a cloud-based repository for Docker images, providing a central location for developers to store, share, and collaborate on container images. It is a key component of the Docker ecosystem, enabling developers to quickly and easily access pre-built images and share their own images with others.

Docker Hub provides a wide range of features and tools for managing Docker images, including:

- **Image Repository**: Docker Hub provides a repository for Docker images, allowing developers to store and share container images with others. It provides a searchable catalog of images, making it easy to find and use the images that you need.
- **User Accounts**: Docker Hub requires users to create an account in order to access its features. User accounts allow developers to manage their own images, as well as contribute to and collaborate on images created by others.
- **Automated Builds**: Docker Hub provides a feature called Automated Builds, which allows developers to automatically build Docker images from source code repositories such as GitHub or Bitbucket. This makes it easy to keep images up-to-date with the latest code changes.
- **Image Tags**: Docker Hub allows developers to tag images with version numbers or other labels, making it easy to manage and track different versions of an image.
- **Teams and Organizations**: Docker Hub allows developers to create teams and organizations, enabling collaboration on Docker images and providing fine-grained access control for different users and groups.
- **Webhooks**: Docker Hub provides a feature called Webhooks, which allows developers to trigger automated actions when certain events occur, such as when a new image is pushed or when a new tag is created.
- **Integration with Other Tools**: Docker Hub integrates with a wide range of other tools and platforms, including GitHub, Bitbucket, Jenkins, and many others. This makes it easy to incorporate Docker Hub into your existing development and deployment workflows.

Docker Hub is a powerful and flexible platform for managing Docker images, providing a central location for storing, sharing, and collaborating on container images. Its features and tools make it easy to manage images, automate builds, and integrate with other tools and platforms, making it an essential component of the Docker ecosystem.

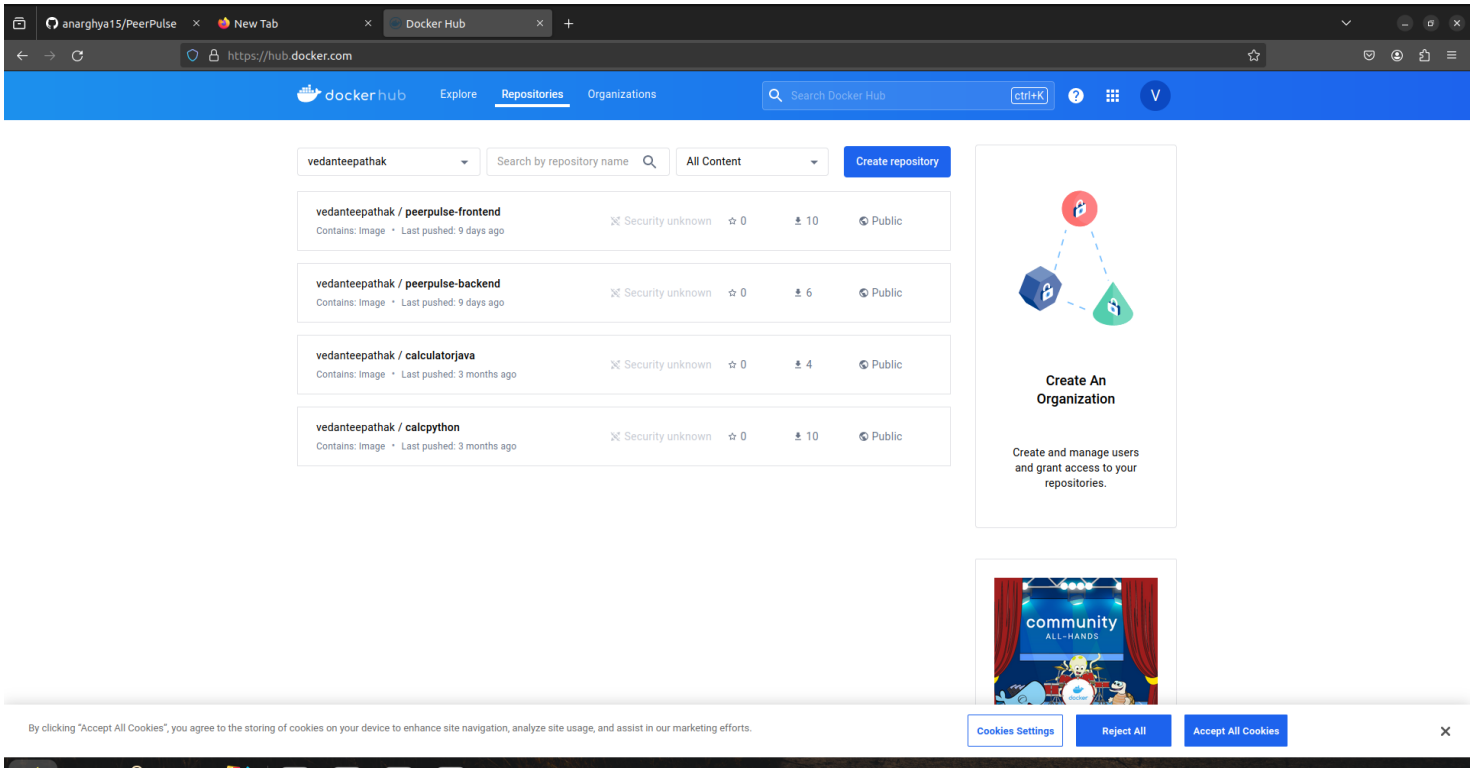
Docker Hub is a cloud-based repository for Docker images, providing a central location for developers to store, share, and collaborate on container images. There are several advantages of using Docker Hub for managing Docker images, including:

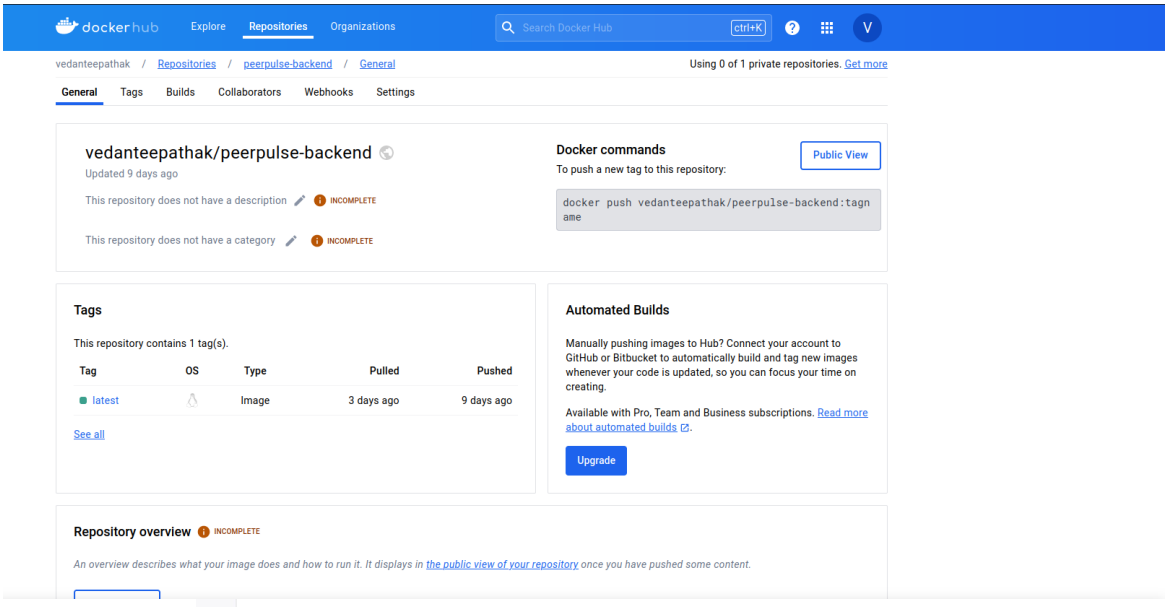
- **Centralized Repository**: Docker Hub provides a centralized repository for Docker images, making it easy to find and access images that have been shared by other developers. This reduces the time and effort required to find and download Docker images, allowing developers to focus on building and deploying their applications.
- **Automated Builds**: Docker Hub provides a feature called Automated Builds, which allows developers to automatically build Docker images from source code repositories such as GitHub or Bitbucket. This makes it easy to keep images up-to-date with the latest code changes and ensures that images are built consistently and reliably.

- **Version Control**: Docker Hub allows developers to version their Docker images, making it easy to track changes and roll back to previous versions if needed. This provides a level of control and visibility over the Docker image lifecycle, ensuring that images are managed effectively and efficiently.
- **Collaboration**: Docker Hub enables developers to collaborate on Docker images, allowing teams to work together to build and maintain images. This promotes knowledge sharing and reduces the amount of duplicate work required, leading to more efficient and effective software development.
- **Security**: Docker Hub provides security features such as image scanning and vulnerability detection, helping to protect against malicious attacks and vulnerabilities. This provides an additional layer of protection for Docker images and ensures that they are safe and secure to use.
- **Flexibility**: Docker Hub supports a wide range of Docker images, including official images from Docker and third-party images from the community. This provides developers with a high degree of flexibility and choice when it comes to selecting and using Docker images.
- **Integration with Other Tools**: Docker Hub integrates with a wide range of other tools and platforms, including GitHub, Bitbucket, Jenkins, and many others. This makes it easy to incorporate Docker Hub into existing development and deployment workflows and ensures that Docker images can be easily integrated into different software systems.

Docker Hub provides a range of advantages for managing Docker images, including centralized storage, automated builds, version control, collaboration, security, flexibility, and integration with other tools. By using Docker Hub, developers can ensure that their Docker images are managed effectively and efficiently, enabling them to focus on building and deploying their applications.

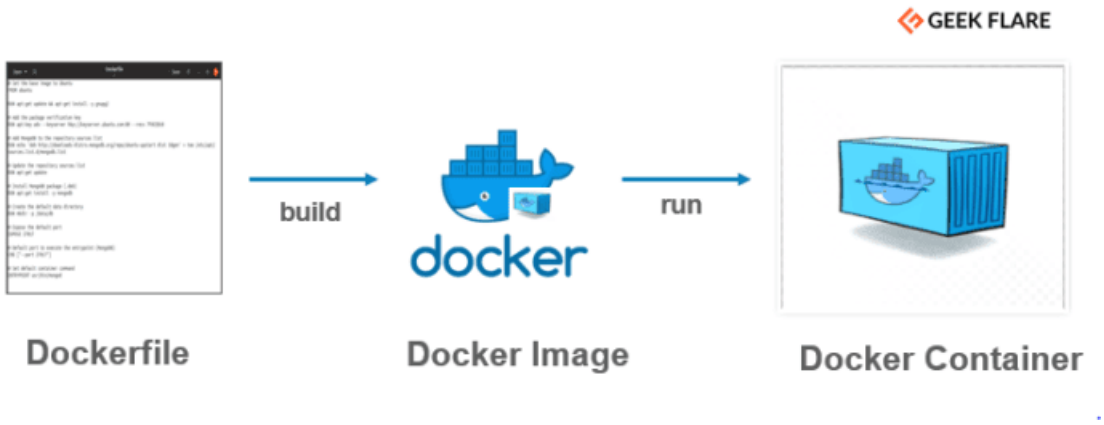
We have two dockerhub images, peerpulse-frontend and peerpulse-backend, correspeonding to the frontend and backend of our application respectively.






## 10.3 Dockerfile

Dockerfile is a text file that contains a set of instructions for building a Docker image. It is used to automate the image creation process and ensure that the resulting image is consistent and repeatable across different environments and platforms.




A Dockerfile typically consists of a series of commands and instructions that specify how to build a Docker image. Some of the most common commands and instructions used in a Dockerfile include:

- `FROM`: This command specifies the base image to use for the new image. The new image will be built on top of the base image, which provides the initial file system and environment for the image.
- `RUN`: This command executes a command in the container during the build process. It is typically used to install software packages or perform other setup tasks.
- `COPY`: This command copies files from the host system into the container. It is typically used to copy application code or configuration files into the container.
- `WORKDIR`: This command sets the working directory for subsequent commands in the Dockerfile. It is typically used to specify the location of application code or other files that need to be accessed during the build process.
- `EXPOSE`: This command specifies the ports that the container will listen on at runtime. It does not actually publish the ports, but instead serves as documentation for users who want to know which ports to use when running the container.

- `CMD`: This command specifies the default command to run when the container is started. It is typically used to start the application or process that the container is designed to run.

Dockerfiles can be used to create images for a wide range of applications and services, including web applications, databases, and microservices. They provide a standardized way to build, package, and deploy applications in a consistent and repeatable manner, making it easier to manage and scale complex software systems.

In addition to the basic commands and instructions discussed above, Dockerfiles can also include advanced features and options, such as multi-stage builds, caching, and environment variables. These features allow developers to optimize the build process and customize images for specific environments and use cases.

Dockerfile is a powerful and flexible tool for building Docker images, providing a standardized and automated way to create images for a wide range of applications and services. By using Dockerfile, developers can ensure that their images are consistent and repeatable, making it easier to manage and scale complex software systems.

Dockerfiles are a powerful and flexible tool for building Docker images, providing a standardized and automated way to create images for a wide range of applications and services. Some of the key advantages of using Dockerfiles include:

- **Standardization**: Dockerfiles provide a standardized way to build Docker images, ensuring that the resulting images are consistent and repeatable across different environments and platforms. This reduces the risk of compatibility issues and makes it easier to manage and deploy software applications.
- **Automation**: Dockerfiles automate the image creation process, reducing the amount of manual work required to build and maintain Docker images. This saves time and resources and reduces the risk of human error.
- **Customization**: Dockerfiles can be customized to meet the specific needs of different applications and environments. Developers can include specific software packages, configuration files, and other components in Dockerfiles to ensure that the resulting images are tailored to their needs.
- **Version Control**: Dockerfiles can be versioned and managed using version control systems such as Git. This provides a history of changes to the Dockerfile and makes it easier to roll back to previous versions if needed.
- **Reusability**: Dockerfiles can be reused across different projects and environments, making it easier to manage and scale software systems. This reduces the amount of duplicate work required and increases the efficiency of the development process.
- **Collaboration**: Dockerfiles can be shared and collaborated on by multiple developers, enabling teams to work together on building and maintaining Docker images. This promotes collaboration and knowledge sharing, leading to more efficient and effective software development.
- **Security**: Dockerfiles can be used to ensure that images are built from trusted sources and that only necessary software packages and components are included in the image. This reduces the attack surface and helps to protect against vulnerabilities and security threats.

Dockerfiles provide a range of advantages for building and managing Docker images, including standardization, automation, customization, version control, reusability, collaboration, and security. By using Dockerfiles, developers can create consistent and repeatable images that are tailored to their needs, making it easier to manage and scale complex software systems.

### 10.3.1 Dockerfiles for the Application

```
FROM openjdk:17
EXPOSE 8500
ADD target/peerpulse-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]

```

Figure 25: Dockerfile for the backend

This is a Dockerfile that creates a Docker image based on the OpenJDK 17 image and adds a JAR file to it. The JAR file is then set as the entry point, which means that when the Docker container is started, it will execute the JAR file using the Java Virtual Machine (JVM).

The `FROM` command specifies the base image for this Docker image, which is OpenJDK 17. This means that this Docker image will have Java 17 installed and available to use.

The `EXPOSE` command informs Docker that the container will listen on the specified network port at runtime. In this case, it is exposing port 8500, which means that any services running inside the container that are listening on port 8500 will be accessible from outside the container.

The `ADD` command copies the JAR file from the build directory to the Docker image. The target directory for the JAR file is `/app.jar`. This means that the JAR file will be copied to the root directory of the container with the name `app.jar`.

The `ENTRYPOINT` command specifies the command that will be executed when the container starts. In this case, it is running the java command with the `-jar` option and passing in the `app.jar` file as an argument. This means that the JAR file will be executed using the Java Virtual Machine (JVM) when the container starts.

```
FROM node:18.13.0 as builder
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 4200
RUN npm run build --prod
RUN ls

FROM nginx:latest
COPY --from=builder /app/dist/peerpulse-frontend /usr/share/nginx/html
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Figure 26: Dockerfile for the frontend

This is a Dockerfile that describes a multi-stage build process for a web application built with Node.js and Angular.

The first stage uses the official Node.js 18.13.0 Docker image as a base image and sets the working directory to `/app`. It then copies all files from the host machine to the container's `/app`

directory. After that, it installs the application's dependencies using `npm install`. The `EXPOSE` 4200 command exposes port 4200, which is the default port used by the Angular development server.

The second stage uses the official Nginx Docker image as a base image. It copies the built files from the first stage to the `/usr/share/nginx/html` directory, which is the default directory used by Nginx to serve static files. It then removes the default Nginx configuration file `/etc/nginx/conf.d/default.conf` and replaces it with a custom configuration file `nginx/nginx.conf` that is copied from the host machine to the container's `/etc/nginx/conf.d` directory.

Finally, the `EXPOSE` 80 command exposes port 80, which is the default port used by HTTP traffic. The `CMD` command starts the Nginx server and runs it in the foreground with the "daemon off;" option, which prevents Nginx from running in the background and keeps the container running.

## 10.4 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define the services that make up an application in a YAML file and then start and stop those services with a single command. Docker Compose is designed to simplify the process of managing multi-container applications, allowing developers to focus on building and deploying their applications rather than managing the infrastructure.


The key features of Docker Compose include:

- **Multi-Container Applications**: Docker Compose is designed for managing multi-container applications. It allows developers to define the services that make up an application, such as a web server, a database, and a caching layer, and then manage those services as a single unit.
- **Declarative Syntax**: Docker Compose uses a declarative YAML syntax for defining the services that make up an application. This makes it easy to define and manage complex applications, as developers can specify the services that they need and let Docker Compose handle the details of starting and stopping those services.
- **Service Dependencies**: Docker Compose allows developers to define dependencies between services, ensuring that services are started in the correct order and that they are able to communicate with each other. This simplifies the process of managing complex applications and reduces the risk of errors and downtime.

- **Scaling**: Docker Compose allows developers to scale services up or down as needed, making it easy to handle changes in traffic or demand. This ensures that applications are able to handle spikes in traffic and that resources are used efficiently.
- **Environment Variables**: Docker Compose allows developers to define environment variables for their services, making it easy to configure applications for different environments. This simplifies the process of managing deployments and reduces the risk of errors caused by misconfiguration.
- **Networking**: Docker Compose allows developers to define custom networks for their application, making it easy to manage communication between services. This ensures that services are able to communicate with each other securely and efficiently.
- **Integration with Other Tools**: Docker Compose integrates with a wide range of other tools and platforms, including Docker Swarm, Kubernetes, and CI/CD tools. This makes it easy to incorporate Docker Compose into existing workflows and ensures that applications can be deployed and managed using a standardized approach.

Docker Compose is a powerful tool for managing multi-container Docker applications. It simplifies the process of defining, configuring, and managing applications, making it easier for developers to focus on building and deploying their applications. With its declarative syntax, support for service dependencies and scaling, and integration with other tools, Docker Compose is a valuable tool for any organization that is building and deploying complex Docker applications.

There are several advantages of using Docker Compose to manage multi-container Docker applications:

- **Simplified Management**: Docker Compose allows developers to define and manage multi-container applications as a single unit, making it easier to manage and deploy complex applications. It simplifies the process of configuring and managing containers, reducing the risk of errors and downtime.
- **Declarative Syntax**: Docker Compose uses a declarative syntax for defining multi-container applications, making it easy to define and manage complex applications. The declarative syntax allows developers to specify the desired state of their application, and Docker Compose handles the details of starting and stopping containers to achieve that state.
- **Service Dependencies**: Docker Compose allows developers to define dependencies between services, ensuring that services are started in the correct order and that they are able to communicate with each other. This simplifies the process of managing complex applications and reduces the risk of errors and downtime.
- **Scaling**: Docker Compose allows developers to scale services up or down as needed, making it easy to handle changes in traffic or demand. This ensures that applications are able to handle spikes in traffic and that resources are used efficiently.
- **Environment Variables**: Docker Compose allows developers to define environment variables for their services, making it easy to configure applications for different environments. This simplifies the process of managing deployments and reduces the risk of errors caused by misconfiguration.
- **Networking**: Docker Compose allows developers to define custom networks for their applications, making it easy to manage communication between services. This ensures that services are able to communicate with each other securely and efficiently.

- **Reproducibility**: Docker Compose makes it easy to reproduce multi-container applications in different environments, ensuring that applications behave consistently across development, testing, and production environments. This reduces the risk of errors caused by differences in environment configuration.
- **Interoperability**: Docker Compose is designed to work seamlessly with other Docker tools and platforms, such as Docker Swarm and Kubernetes. This ensures that applications can be deployed and managed using a standardized approach, regardless of the underlying infrastructure.
- **Collaboration**: Docker Compose enables teams to collaborate on the definition and management of multi-container applications, promoting knowledge sharing and reducing the amount of duplicate work required.

Docker Compose provides a powerful and flexible tool for managing multi-container Docker applications. Its simplified management, declarative syntax, support for service dependencies and scaling, and integration with other tools make it an essential tool for any organization that is building and deploying complex Docker applications.

### 10.4.1 Installation

Here are the step-by-step instructions to install Docker Compose on Ubuntu:

1. **Update the package index**:

   Before installing any new software, it is always a good practice to ensure that your Ubuntu system is up to date. Run the following command to update the package index:

   ```
   sudo apt-get update
   ```

   This will download the latest package information from the Ubuntu repositories.

2. **Install Docker**:

   Docker Compose is built on top of the Docker platform, so you need to have Docker installed on your Ubuntu system. If you haven't installed Docker already, you can do so with the following command:

   ```
   sudo apt-get install docker.io
   ```

   This will install the Docker engine on your system.

3. **Download Docker Compose**:

   Once you have Docker installed, you need to download the Docker Compose binary. You can do this with the following command:

   ```
   sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/
   docker-compose-\$(uname -s)-\$(uname -m)" -o /usr/local/bin/docker-compose
   ```

   This command downloads the latest version of Docker Compose (replace "1.29.2" with the latest version) and saves it to the `/usr/local/bin` directory.

4. **Make Docker Compose executable**:

   The downloaded Docker Compose binary needs to be made executable before you can use it. Run the following command to do this:

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

   This command sets the execute permission on the downloaded binary.

   Verify Docker Compose installation

To verify that Docker Compose has been installed correctly, run the following command:

```
docker-compose -version
```

This command should display the version number of Docker Compose that was installed on your system.

We have successfully installed Docker Compose on your Ubuntu system. We can now use it to manage multi-container Docker applications.

### 10.4.2 Basic Commands

Docker Compose is a tool for defining and running multi-container Docker applications. Here are some of the most commonly used Docker Compose commands:

- **docker-compose up**: This command is used to start a Docker Compose application. The syntax for this command is:

  ```
  docker-compose up <services>
  ```

  You can also specify specific services to start by listing them after the up command.

- **docker-compose down**: This command is used to stop a Docker Compose application and remove the containers. The syntax for this command is:

  ```
  docker-compose down
  ```

- **docker-compose ps**: This command is used to list the containers in a Docker Compose application. The syntax for this command is:

  ```
  docker-compose ps
  ```

  This command lists the containers and their status, as well as the ports they are mapped to.

- **docker-compose build**: This command is used to build the Docker images for the services defined in a Docker Compose file. The syntax for this command is:

  ```
  docker-compose build <services>
  ```

  For example, to build the images for the "web" and "db" services, you would run:

  ```
  docker-compose build web db
  ```

- **docker-compose logs**: This command is used to view the logs for a Docker Compose application. The syntax for this command is:

  ```
  docker-compose logs <services>
  ```

- **docker-compose exec**: This command is used to execute a command in a running container. The syntax for this command is:

  ```
  docker-compose exec <service> <command>
  ```

- **docker-compose config**: This command is used to validate and view the Docker Compose configuration. The syntax for this command is:

  ```
  docker-compose config
  ```

  This command validates the "docker-compose.yml" file and displays the merged configuration.

- **docker-compose up –build**: This command is used to rebuild the Docker images before starting the Docker Compose application. The syntax for this command is:

  `docker-compose up -build`

  This command rebuilds the Docker images for all services before starting the containers.

These are just a few examples of the many Docker Compose commands available. Each command has its own set of options and flags, so be sure to consult the documentation for more information.

### 10.4.3 Docker Compose for the Application

**Docker Compose was used by us in the initial phases of testing and deploying our application locally.**

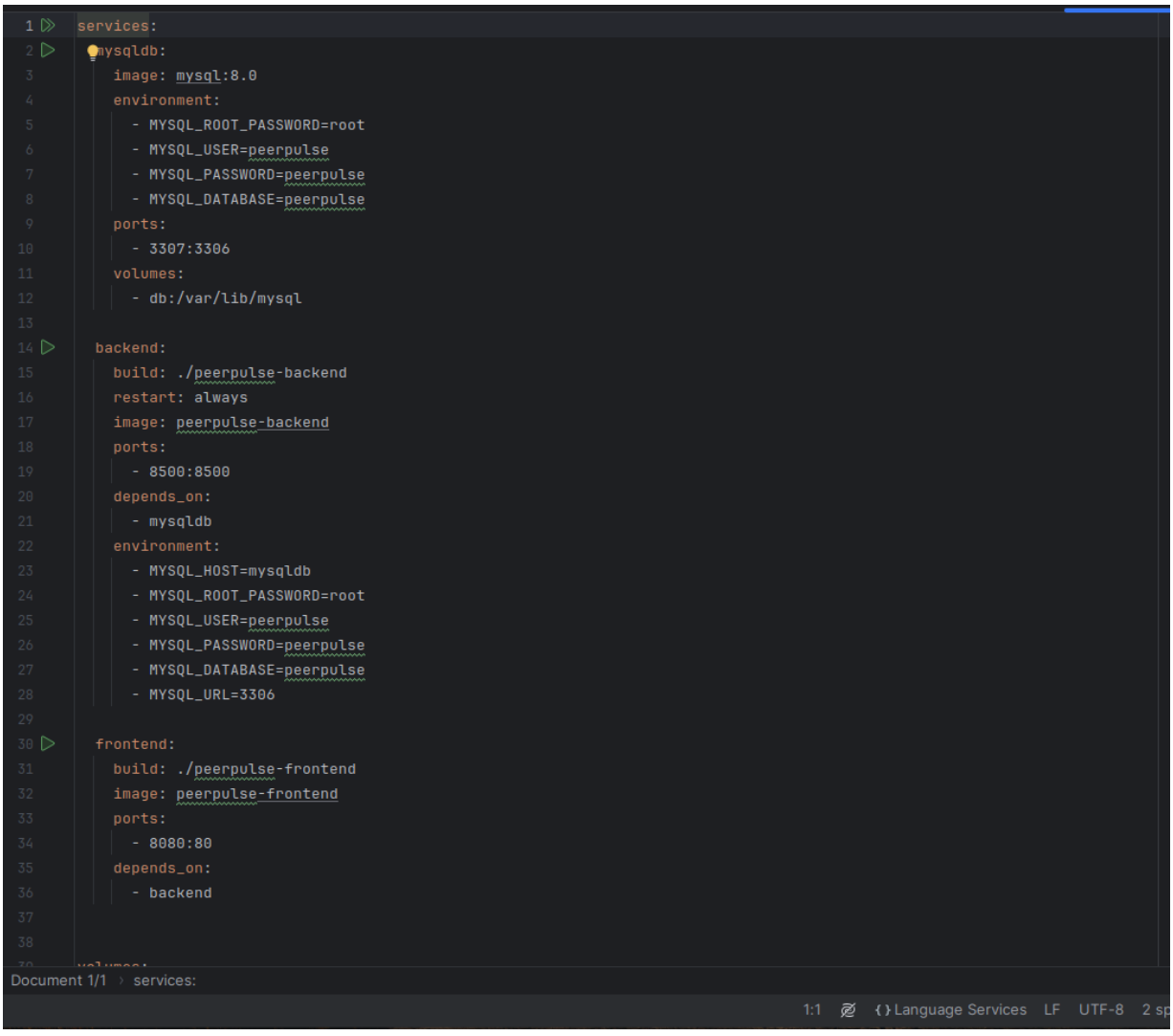



Figure 27: A snippet of the Docker Compose

This is a docker-compose file that defines three services: a MySQL database, a backend server, and a frontend server.

The first service, `mysqldb`, uses the `mysql:5.7` image and sets environment variables for the root password, a MySQL user, password, and database name. The service also maps port 3306 in the container to port 3307 on the host machine and uses a volume named `db` to persist data.

The second service, `backend`, builds an image from a Dockerfile located in `./peerpulse-backend` sets environment variables for the MySQL hostname, root password, user, password, database name, and URL, and depends on the `mysqldb` service. The service maps port 8500 in the container to port 8500 on the host machine and restarts always.

The third service, `frontend`, builds an image from a Dockerfile located in `./peerpulse-frontend` and maps port 4200 in the container to port 4200 on the host machine.

Finally, a volume named `db` is defined to store the MySQL data.

## 11 Container Orchestration and Deployment

Container orchestration is the process of automating the deployment, scaling, and management of containerized applications. It allows organizations to manage large and complex container environments, ensuring that applications are deployed and run consistently and reliably across different infrastructure and environments.

Container orchestration is necessary because modern applications are often composed of multiple services, each running in its own container. These containers need to be deployed and managed in a way that ensures they work together seamlessly and can be scaled up or down as needed.

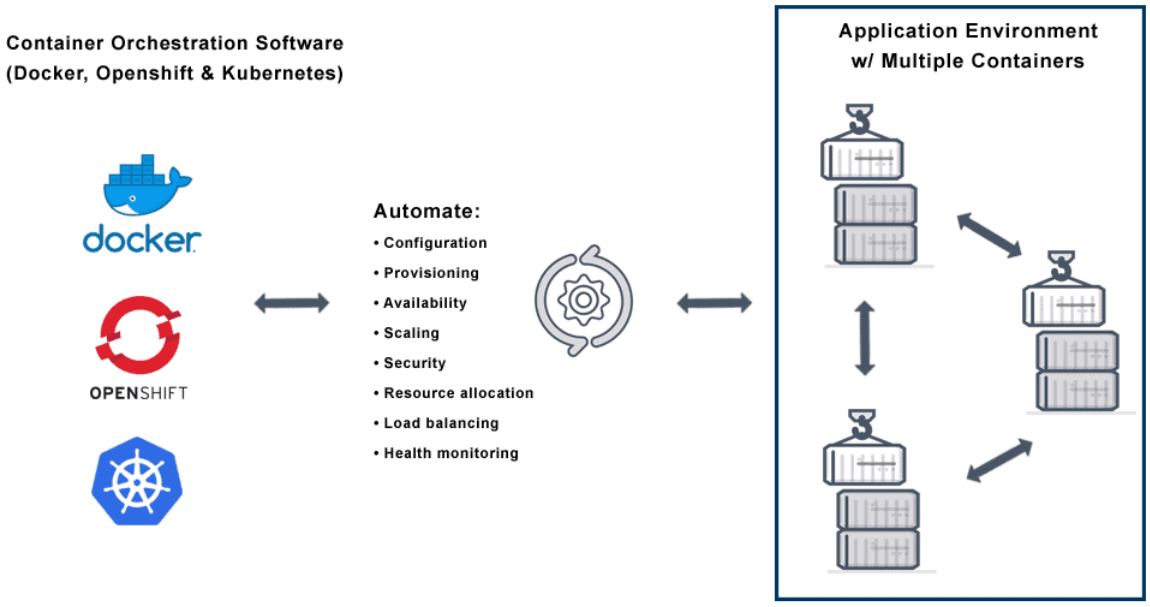



Figure 28: A overview of container orchestration

Container orchestration platforms provide a range of features and tools that enable organizations to manage containerized applications at scale. These features include:

- **Service discovery**: Container orchestration platforms provide a mechanism for services to discover each other and communicate with each other. This is typically done through a service discovery mechanism, such as DNS or a load balancer.
- **Load balancing**: Container orchestration platforms provide load balancing capabilities that allow traffic to be distributed across multiple instances of a service.

- **Scaling**: Container orchestration platforms provide the ability to scale services up or down based on demand. This can be done manually or automatically, based on predefined rules and thresholds.
- **High availability**: Container orchestration platforms provide mechanisms for ensuring high availability, such as automatic failover and replication.
- **Rolling updates**: Container orchestration platforms provide the ability to perform rolling updates, which allow new versions of a service to be deployed gradually, without disrupting the overall system.
- **Health monitoring**: Container orchestration platforms provide health monitoring capabilities that allow services to be monitored for errors or failures. This allows issues to be detected and addressed before they become critical.
- **Configuration management**: Container orchestration platforms provide a way to manage the configuration of containers and services, ensuring that they are deployed with the correct settings.
- **Security**: Container orchestration platforms provide a range of security features, such as authentication and authorization, that ensure that containers and services are secure and compliant.

There are several popular container orchestration platforms available, including Kubernetes, Docker Swarm, and Apache Mesos. Each platform has its own strengths and weaknesses, and the choice of platform will depend on the specific needs of the organization and the applications being deployed.

Container orchestration is a critical component of modern application development and deployment. It allows organizations to manage containerized applications at scale, ensuring that they are deployed and run consistently and reliably across different infrastructure and environments. Container orchestration platforms, such as Kubernetes, provide a range of features and tools that make it easy to manage large and complex container environments, and are widely used in production environments.

## 11.1 Kubernetes

Kubernetes is an open-source container orchestration platform developed by Google. It provides a way to automate the deployment, scaling, and management of containerized applications. Kubernetes is designed to be portable and extensible, and can be deployed on a wide range of infrastructure, from on-premises data centers to public cloud providers.

Kubernetes provides a rich set of features and tools for managing containerized applications, including:

- **Pods**: Kubernetes uses pods as the smallest deployable units in a cluster. A pod contains one or more containers, and provides a way to group containers together and manage their lifecycle.
- **Services**: Kubernetes provides a way to expose pods to the network, using services. Services provide a stable IP address and DNS name for a set of pods, and allow traffic to be load balanced across them.
- **Replication Controllers**: Kubernetes provides a way to ensure that a specified number of replicas of a pod are running at all times. This is done using replication controllers, which monitor the state of pods and ensure that the desired number of replicas are running.

- **Deployments**: Kubernetes provides a way to manage the deployment of new versions of an application, using deployments. Deployments allow new versions of an application to be rolled out gradually, without disrupting the overall system.
- **ConfigMaps and Secrets**: Kubernetes provides a way to manage configuration data, using ConfigMaps and Secrets. ConfigMaps allow configuration data to be stored as key-value pairs, while Secrets allow sensitive data to be stored securely.
- **StatefulSets**: Kubernetes provides a way to manage stateful applications, using StatefulSets. StatefulSets provide a way to ensure that pods are deployed and scaled in a predictable and consistent manner.
- **DaemonSets**: Kubernetes provides a way to run a single copy of a pod on each node in a cluster, using DaemonSets. DaemonSets can be used to deploy cluster-level services, such as log collectors or monitoring agents.

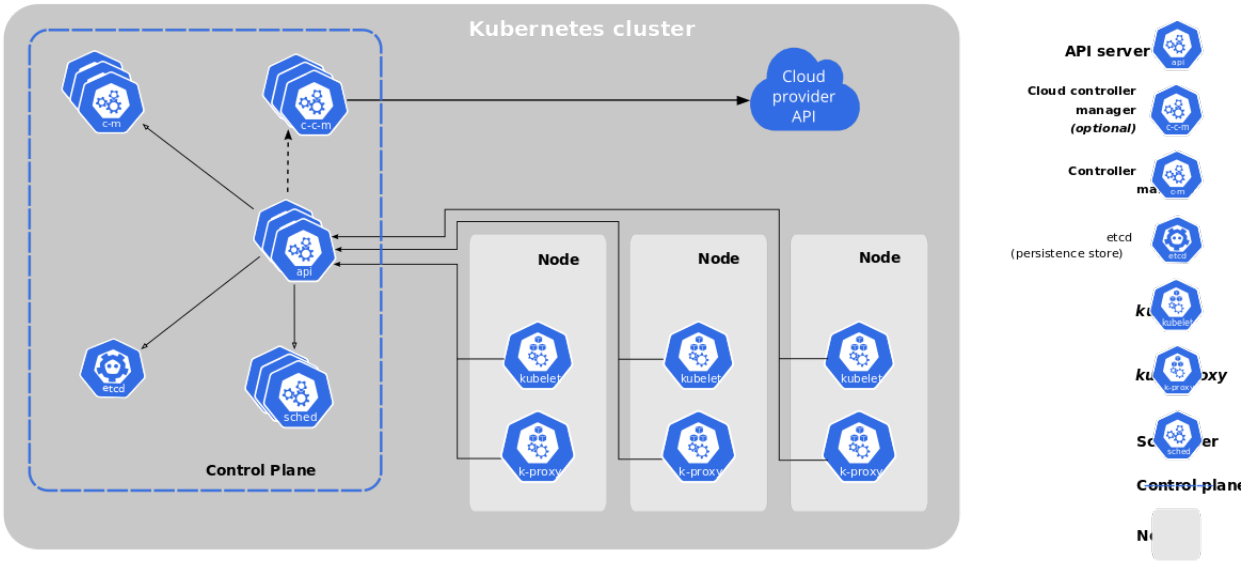



Figure 29: The components of Kubernetes

Kubernetes also provides a range of features and tools for managing the underlying infrastructure and resources, including:

- **Nodes**: Kubernetes uses nodes as the underlying infrastructure for running containers. A node can be a physical or virtual machine, and runs the Kubernetes agent, known as the kubelet.
- **Pods**: Kubernetes uses pods as the basic building block for running containers. Each pod runs one or more containers, and provides a shared network namespace and storage volume.
- **Services**: Kubernetes provides a way to expose pods to the network, using services. Services provide a stable IP address and DNS name for a set of pods, and allow traffic to be load balanced across them.
- **Volumes**: Kubernetes provides a way to manage persistent storage, using volumes. Volumes can be used to store data that needs to persist across container restarts, such as databases or file systems.

- **Namespaces**: Kubernetes provides a way to organize resources and control access, using namespaces. Namespaces allow resources to be partitioned and isolated, and provide a way to control access to those resources.
- **Resource quotas**: Kubernetes provides a way to limit resource usage, using resource quotas. Resource quotas allow administrators to limit the amount of CPU, memory, and storage that can be used by a particular namespace or user.
- **Horizontal Pod Autoscaler**: Kubernetes provides a way to automatically scale the number of pods in a deployment based on demand, using the Horizontal Pod Autoscaler (HPA). The HPA uses metrics such as CPU usage or custom metrics to determine when to scale up or down the number of pods.

Kubernetes is designed to be highly available and scalable, and can be used to manage large and complex containerized environments. It is also designed to be portable and extensible, and can be customized and extended using a wide range of plugins and extensions.

Kubernetes has become the de facto standard for container orchestration, and is widely used in production environments. It is supported by a large and active community of contributors, and is used by a wide range of organizations, from startups to global enterprises.

Some of the benefits of using Kubernetes include:

- **Scalability**: Kubernetes provides a way to scale applications and services up or down based on demand, ensuring that resources are used efficiently and cost-effectively.
- **Reliability**: Kubernetes provides a way to ensure that applications and services are highly available and resilient to failures, ensuring that users can access them when they need to.
- **Flexibility**: Kubernetes provides a way to deploy and manage containerized applications across a wide range of infrastructure and environments, from on-premises data centers to public cloud providers.
- **Portability**: Kubernetes provides a way to deploy and manage containerized applications in a consistent and portable manner, ensuring that they can be easily moved between different infrastructure and environments.
- **Extensibility**: Kubernetes provides a way to customize and extend the platform using a wide range of plugins and extensions, allowing organizations to tailor the platform to their specific needs and requirements.

Kubernetes is a powerful and flexible container orchestration platform that provides a way to automate the deployment, scaling, and management of containerized applications. Its rich set of features and tools, combined with its scalability, reliability, and portability, make it a popular choice for organizations of all sizes and industries.

### 11.1.1 Installation

Here's a step-by-step guide on how to install Kubernetes on Ubuntu:

1. Update your Ubuntu system:

   ```
   sudo apt update && sudo apt upgrade
   ```

2. Install Docker:

   ```
   sudo apt install docker.io
   ```

3. Add the Kubernetes apt repository key:

   ```
   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
   ```

4. Add the Kubernetes apt repository:

   ```
   sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
   ```

5. Update your Ubuntu system again:

   ```
   sudo apt update
   ```

6. Install kubelet, kubeadm, and kubectl:

   ```
   sudo apt install kubelet kubeadm kubectl
   ```

7. Disable swap on your Ubuntu system:

   ```
   sudo swapoff -a
   ```

8. Initialize the Kubernetes cluster:

   ```
   sudo kubeadm init -pod-network-cidr=10.244.0.0/16
   ```

   Note: The "`-pod-network-cidr`" option specifies the range of IP addresses that will be used for Kubernetes pods.

9. Copy the Kubernetes configuration file to your home directory:

   ```
   mkdir -p $HOME/.kube sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   sudo chown $(id -u):$(id -g) $HOME/.kube/config
   ```

10. Install a network add-on for Kubernetes:

    ```
    kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
    ```

11. Verify that the Kubernetes cluster is up and running:

    ```
    kubectl get nodes
    ```

    This should display the list of nodes in your Kubernetes cluster.

We have successfully installed Kubernetes on Ubuntu.

### 11.1.2 Basic Commands

Kubectl is the command-line tool used to interact with a Kubernetes cluster. It allows users to create, modify, and manage Kubernetes resources like Pods, Deployments, Services, and ConfigMaps. Here are some of the most commonly used kubectl commands:

- **kubectl create**: This command is used to create a new Kubernetes resource. For example, to create a new deployment, you would run:

  ```
  kubectl create deployment my-deployment -image=my-image
  ```

- **kubectl get**: This command is used to retrieve information about Kubernetes resources. For example, to get a list of all the pods in the default namespace, you would run:

  ```
  kubectl get pods
  ```

- **kubectl describe**: This command is used to get detailed information about a specific Kubernetes resource. For example, to get detailed information about a pod named mypod, you would run:

  ```
  kubectl describe pod my-pod
  ```

- **kubectl delete**: This command is used to delete a Kubernetes resource. For example, to delete a deployment named my-deployment, you would run:

  ```
  kubectl delete deployment my-deployment
  ```

- **kubectl apply**: This command is used to apply a configuration file to a Kubernetes cluster. For example, to create a deployment using a YAML file named `my-deployment.yaml`, you would run:

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

- **kubectl logs**: This command is used to view the logs of a specific container in a pod. For example, to view the logs of the container named `my-container` in a pod named `my-pod`, you would run:

  ```
  kubectl logs my-pod -c my-container
  ```

- **kubectl exec**: This command is used to execute a command inside a container in a pod. For example, to execute the command ls inside a container named `my-container` in a pod named `my-pod`, you would run:

  ```
  kubectl exec my-pod -c my-container - ls
  ```

- **kubectl port-forward**: This command is used to forward a port from a Kubernetes pod to the local machine. For example, to forward port 8080 from a pod named my-pod to port 8888 on the local machine, you would run:

  ```
  kubectl port-forward my-pod 8888:8080
  ```

These are just some of the most commonly used kubectl commands. There are many more commands available, each with its own set of options and flags.

### 11.1.3 Kubernetes CLI Plugin

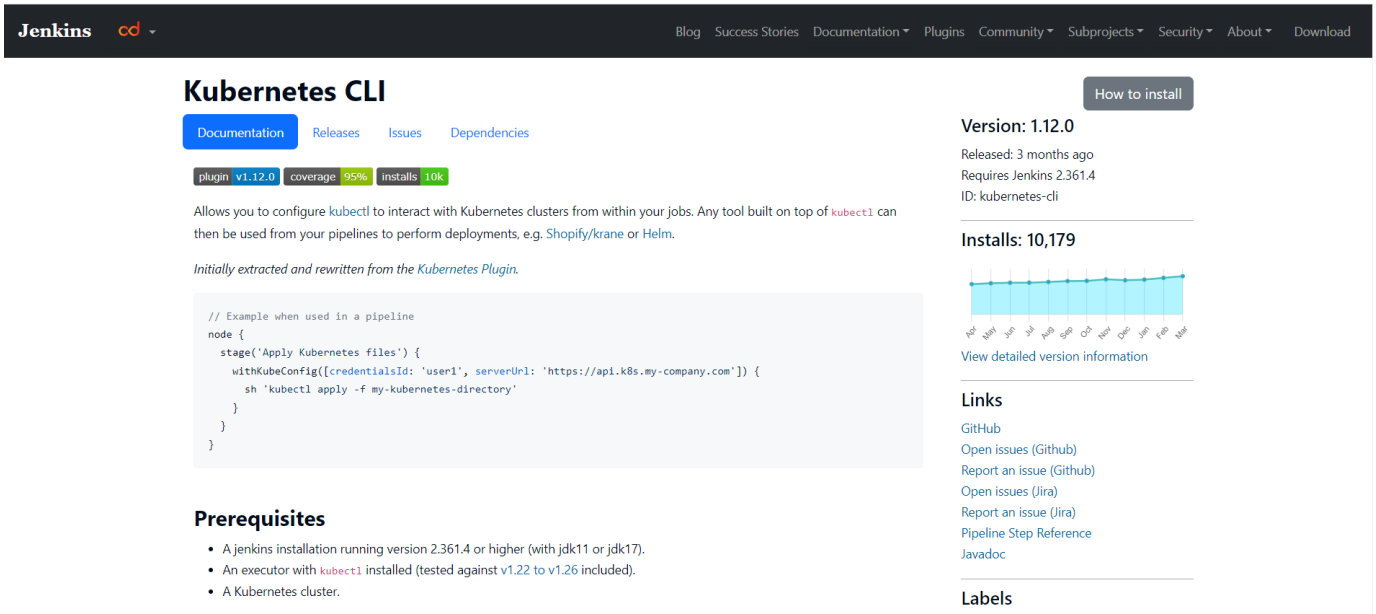
The Kubernetes CLI plugin for Jenkins is a plugin that allows Jenkins to interact with a Kubernetes cluster using the `kubectl` command-line tool. This plugin enables Jenkins to deploy applications and manage resources in a Kubernetes cluster as part of a Jenkins pipeline.

With the Kubernetes CLI plugin installed, Jenkins can execute `kubectl` commands within a pipeline, allowing for easy deployment of applications to the Kubernetes cluster. This plugin also provides a convenient way to manage Kubernetes resources, such as pods, services, and deployments, directly from Jenkins.

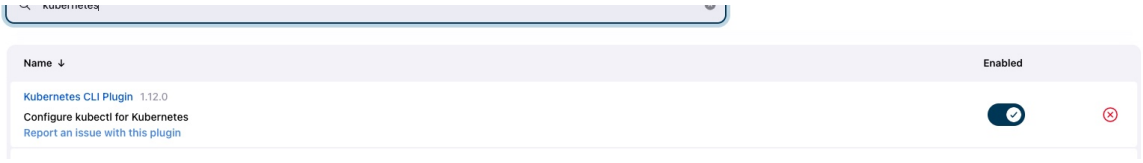
To use the plugin, you will need to configure the Kubernetes cluster credentials in Jenkins. This can be done through the Jenkins web interface or in a Jenkinsfile using the Kubernetes CLI plugin's Pipeline syntax.

Once the credentials are configured, you can use the `kubectl` command in your Jenkins pipeline scripts to interact with your Kubernetes cluster. For example, you can create a deployment by running `kubectl apply -f deployment.yaml`. You can also use the `kubectl` command in a Jenkins shell step to execute arbitrary commands on the Kubernetes cluster.

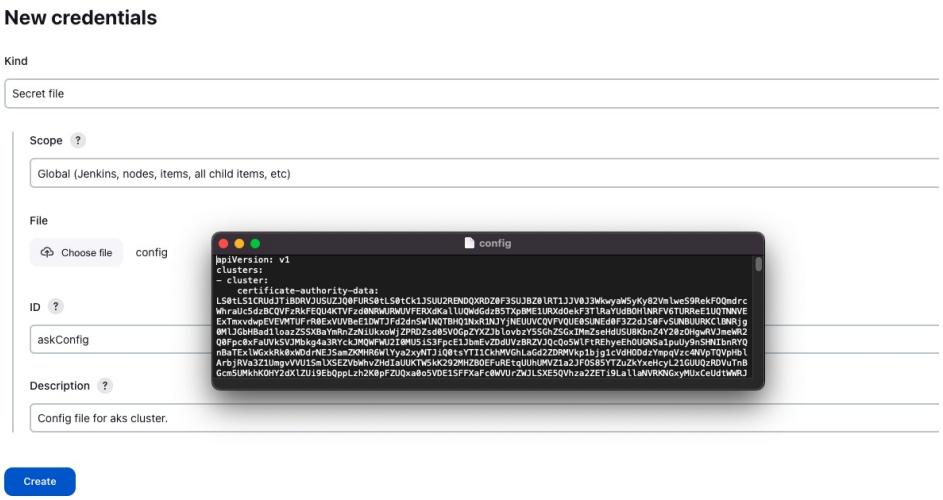
The Kubernetes CLI plugin for Jenkins provides a powerful way to integrate Kubernetes deployment and management into your Jenkins pipelines.




For using the plugin, make sure that it is installed. If not installed, go to manage jenkins > manage plugins > available plugins and install it.




Create credentials in formn of a secret file, which will store the kubeconfig file of the cluster our kube cli is connecting to.




We can use various commands which are dependent on kubectl configuration such as helm. Following figure shows an example of some commands.

```
stage('Deploying logstash and filebeats inside the kubernetes cluster') {
    steps {
        withKubeConfig([credentialsId: 'aksConfig', serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io']) {
            sh 'helm upgrade --install filebeat kubeDeploy/filebeat'
            sh 'helm upgrade --install logstash kubeDeploy/logstash'
        }
    }
}
stage('Adding secrets and config maps to kubernetes cluster') {
    steps {
        withKubeConfig([credentialsId: 'aksConfig', serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io']) {
            sh 'kubectl apply -f kubeDeploy/mysql-root-credentials.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-credentials.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-configmap.yml'
        }
    }
}

stage('Deleting older application deployment') {
    steps {
        withKubeConfig([credentialsId: 'aksConfig', serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io']) {
            sh 'kubectl delete -f kubeDeploy/mysql-deployment.yml'
```

## 11.2 Minikube

**We have used Minikube for local environment deployment testing, before moving on to use Azure Kubernetes Service on the cloud.**

Minikube is a tool that allows developers to run and test Kubernetes clusters locally, on their own machine. It provides a way to test Kubernetes deployments and configurations before deploying them to a production environment. Minikube is designed to be easy to use and lightweight, and can be installed on a wide range of platforms, including Windows, macOS, and Linux.

Minikube provides a simple and easy-to-use interface for creating and managing Kubernetes clusters. It uses a single-node cluster configuration, which means that all Kubernetes components, such as the API server, etcd, and kubelet, run on a single machine. This makes it easy to get started with Kubernetes, without the need for a large and complex infrastructure.

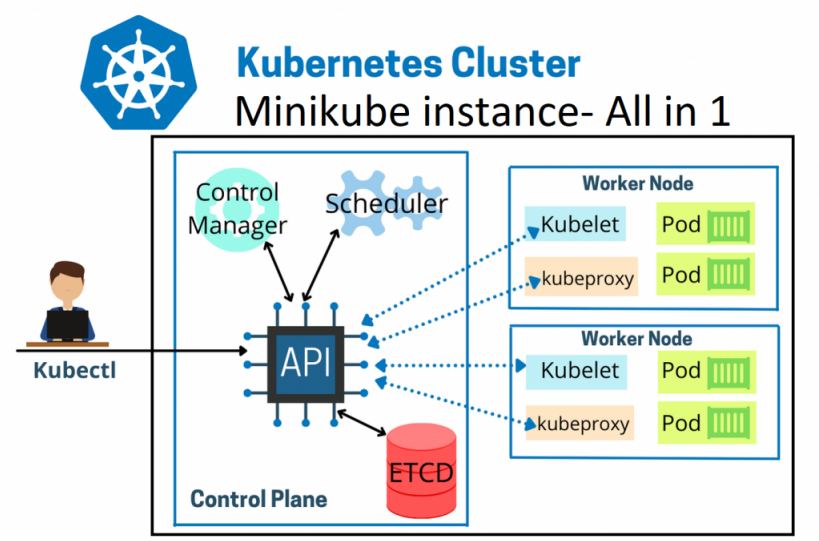



Figure 30: The architecture of Minikube

Minikube provides a range of features and tools for managing Kubernetes clusters, including:

- **Cluster creation**: Minikube provides a way to create a Kubernetes cluster with a single command. The cluster can be customized with a range of configuration options, such as the Kubernetes version, network settings, and addons.
- **Cluster management**: Minikube provides a way to start and stop Kubernetes clusters, as well as to check the status of the cluster and its components. It also provides a way to delete clusters when they are no longer needed.
- **Addons**: Minikube provides a range of addons that can be enabled to extend the functionality of the Kubernetes cluster. Addons include features such as a dashboard, a container registry, and a load balancer.
- **Kubernetes dashboard**: Minikube provides a way to access the Kubernetes dashboard, which provides a graphical interface for managing the Kubernetes cluster.
- **Container runtime**: Minikube supports a range of container runtimes, including Docker, CRI-O, and containerd. This allows developers to test their applications with different container runtimes, and to choose the runtime that best meets their needs.
- **Networking**: Minikube provides a way to configure the networking settings for the Kubernetes cluster, including the pod and service CIDRs, and the network plugin.
- **Storage**: Minikube provides a way to configure the storage settings for the Kubernetes cluster, including the default storage class, and the persistent volume and claim settings.

One of the key benefits of Minikube is its ease of use. It can be downloaded and installed quickly and easily, and can be used to create a local Kubernetes cluster in just a few minutes. This makes it a popular tool for developers who want to test their applications with Kubernetes, without the need for a large and complex infrastructure.

Another benefit of Minikube is its portability. It can be installed on a wide range of platforms, including Windows, macOS, and Linux, and can be used to create Kubernetes clusters that can be easily moved between different environments.

Minikube is also highly configurable, and provides a range of options for customizing the Kubernetes cluster. Developers can choose the Kubernetes version, network settings, and addons, and can configure the cluster to meet their specific needs and requirements.

Minikube is a powerful and flexible tool for running and testing Kubernetes clusters locally. Its ease of use, portability, and configurability make it a popular choice for developers who want to test their applications with Kubernetes, without the need for a large and complex infrastructure.

### 11.2.1 Installation

Here's a step-by-step guide on how to install Minikube on Ubuntu:

1. Install VirtualBox:

   ```
   sudo apt update sudo apt install virtualbox
   ```

2. Install kubectl:

   ```
   sudo apt update sudo apt install -y apt-transport-https gnupg2 curl curl -s
   https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add - echo
   "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.lis
   sudo apt update sudo apt install -y kubectl
   ```

3. Install Minikube:

   ```
   curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
   sudo install minikube-linux-amd64 /usr/local/bin/minikube
   ```

4. Start Minikube:

   ```
   minikube start
   ```

5. Verify that Minikube is running:

   ```
   kubectl get pods -A
   ```

   This should display the list of pods in the default namespace.

We have successfully installed Minikube on Ubuntu and started a local Kubernetes cluster. You can now use Minikube to develop and test your Kubernetes applications.

### 11.2.2 Basic Commands

Here are some of the most commonly used Minikube commands:

- **minikube start**: This command is used to start a local Kubernetes cluster using Minikube. The syntax for this command is:

  ```
  minikube start <flags>
  ```

  For example, to start a cluster with 2 CPUs and 4GB of memory, you would run:

  ```
  minikube start -cpus=2 -memory=4096
  ```

- **minikube stop**: This command is used to stop a running Minikube cluster. The syntax for this command is:

  ```
  minikube stop
  ```

- **minikube delete**: This command is used to delete a Minikube cluster. The syntax for this command is:

  ```
  minikube delete
  ```

- **minikube status**: This command is used to check the status of a running Minikube cluster. The syntax for this command is:

  ```
  minikube status
  ```

- **minikube dashboard**: This command is used to open the Kubernetes dashboard in a web browser. The syntax for this command is:

  ```
  minikube dashboard
  ```

- **minikube ssh**: This command is used to SSH into the Minikube virtual machine. The syntax for this command is:

  ```
  minikube ssh
  ```

- **minikube service**: This command is used to expose a Kubernetes service as a NodePort service. The syntax for this command is:

  ```
  minikube service <service-name>
  ```

  For example, to expose a service named "`my-service`", you would run:

  ```
  minikube service my-service
  ```

- **minikube addons**: This command is used to manage Minikube addons. The syntax for this command is:

  ```
  minikube addons <add-on-name>
  ```

  For example, to enable the "`dashboard`" addon, you would run:

  ```
  minikube addons enable dashboard
  ```

## 11.3 Azure

Microsoft Azure is a cloud computing platform and service offered by Microsoft. It provides a wide range of cloud-based services, including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS).

Azure enables organizations to build, deploy, and manage applications and services from a global network of data centers. It provides a scalable and flexible platform that can be used to power a wide range of applications, from simple web applications to complex enterprise-grade solutions.

Azure offers a wide range of services and features, including:

- **Virtual machines**: Azure provides a range of virtual machine sizes and configurations that can be used to run a wide range of applications and workloads.
- **Storage**: Azure provides a range of storage options, including blob storage, file storage, and disk storage, that can be used to store and manage data.
- **Networking**: Azure provides a range of networking services, including virtual networks, load balancers, and VPN gateways, that can be used to connect applications and services.
- **Databases**: Azure provides a range of database services, including SQL Database, Cosmos DB, and MySQL, that can be used to store and manage data.
- **Identity and access management**: Azure provides a range of identity and access management services, including Azure Active Directory, that can be used to manage user identities and access to resources.
- **DevOps**: Azure provides a range of DevOps services, including Azure DevOps and GitHub, that can be used to manage the software development lifecycle.
- **Internet of Things (IoT)**: Azure provides a range of IoT services, including IoT Hub and Event Hubs, that can be used to connect, monitor, and manage IoT devices.

Azure also provides a range of AI and machine learning services, including Azure Machine Learning and Cognitive Services, that can be used to build intelligent applications.

Azure is also designed to integrate with a wide range of third-party tools and services, making it easy to use with existing tools and workflows. It provides a range of APIs and SDKs that can be used to build custom integrations and applications.

One of the key benefits of Azure is its scalability and flexibility. Organizations can quickly and easily scale their applications and services as demand grows, without the need to invest in additional hardware or infrastructure. Azure also provides a range of pricing options, including pay-as-you-go and reserved instances, that can be used to optimize costs and maximize value.

Another key benefit of Azure is its security and compliance features. Azure provides a range of security and compliance controls, including encryption, access controls, and compliance certifications, that can be used to protect data and ensure regulatory compliance.

In summary, Azure is a powerful cloud computing platform that provides a wide range of services and features for building, deploying, and managing applications and services. Its scalability, flexibility, and integration with third-party tools make it a popular choice for organizations of all sizes and industries.

## 11.4 Azure Kubernetes Service

**Azure Kubernetes Service (AKS) is being used by us for deployment on the cloud.**

Azure Kubernetes Service (AKS) is a managed Kubernetes service offered by Microsoft Azure. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. AKS makes it easy to deploy and operate Kubernetes clusters in Azure, providing a scalable and flexible platform for deploying and managing containerized applications.

AKS provides a range of features and benefits, including:

- **Simplified Kubernetes deployment and management**: AKS provides a fully managed Kubernetes service, which means that Microsoft manages the underlying infrastructure and handles tasks such as upgrades, patches, and scaling. This makes it easy to deploy and manage Kubernetes clusters in Azure, without the need for extensive expertise in Kubernetes.
- **High availability and scalability**: AKS provides a highly available and scalable platform for deploying and managing containerized applications. It uses Azure's global network of data centers to provide high availability and low latency, and it can scale up or down based on demand.
- **Security and compliance**: AKS provides a range of security and compliance features, including network security, authentication and authorization, and compliance certifications. This makes it easy to deploy and manage containerized applications in a secure and compliant manner.
- **Integration with Azure services**: AKS integrates with a wide range of Azure services, including Azure Container Registry, Azure Active Directory, Azure Monitor, and Azure DevOps. This makes it easy to build, deploy, and manage containerized applications using familiar Azure tools and workflows.
- **Cost optimization**: AKS provides a range of cost optimization features, including node auto-scaling, pod auto-scaling, and spot instances. This makes it easy to optimize costs and maximize value, without sacrificing performance or availability.
- **Open-source and community-driven**: AKS is built on open-source technologies and is part of the Kubernetes community. This means that it benefits from ongoing development and innovation from a large and active community of contributors.

AKS also provides a range of tools and features for deploying and managing containerized applications, including:

- **Kubernetes dashboard**: AKS provides a web-based Kubernetes dashboard that allows you to view and manage your Kubernetes clusters, pods, and services.
- **kubectl CLI**: AKS provides the kubectl CLI, which allows you to interact with your Kubernetes clusters from the command line.
- **Azure CLI and PowerShell**: AKS integrates with Azure CLI and PowerShell, allowing you to manage your Kubernetes clusters using familiar Azure tools and workflows.
- **Helm charts**: AKS supports Helm charts, which are a package manager for Kubernetes that allows you to deploy and manage complex applications.
- **Azure DevOps integration**: AKS integrates with Azure DevOps, allowing you to deploy and manage containerized applications using familiar DevOps workflows.

Azure Kubernetes Service provides a powerful and flexible platform for deploying and managing containerized applications in Azure. Its integration with Azure services, security and compliance features, and cost optimization capabilities make it a popular choice for organizations of all sizes and industries.

# 12 Code Walkthrough

The Spring Boot backend is built using Java, and it uses Spring Framework to handle requests and responses. The backend provides RESTful APIs that the Angular frontend can consume.

The Angular frontend is built using TypeScript and Angular Framework. It provides a user interface that allows users to interact with the backend APIs.

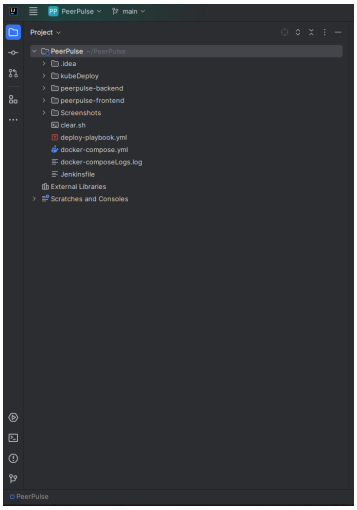



Figure 31: The overall directory structure of the application

## 12.1 Frontend

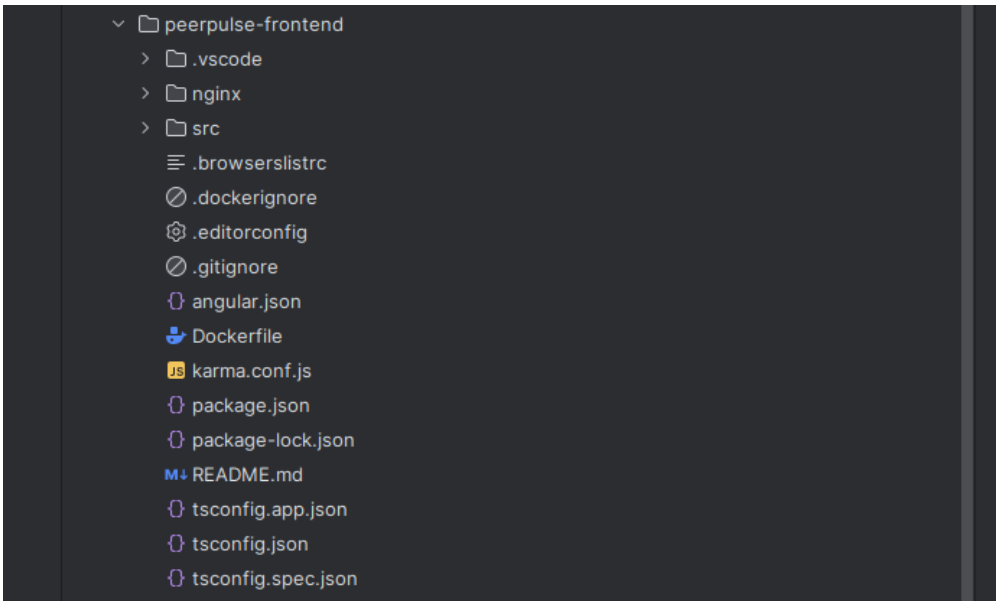



Figure 32: The directory structure of the frontend

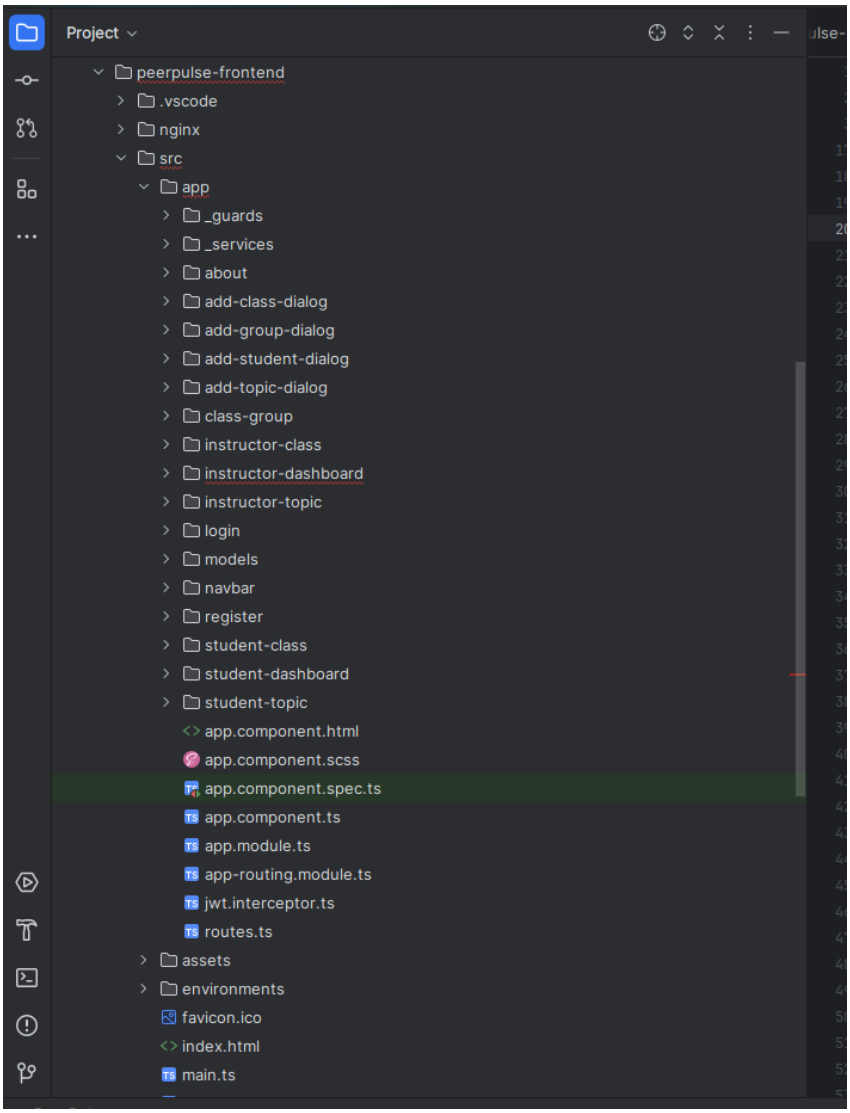



Figure 33: The extended directory structure of the frontend

An Angular project typically consists of a number of different components that work together to create a functional web application. Here is an overview of some of the key components found in the project:

- **App**: This is the main component of your Angular application. It contains the root component for your application and provides a container for other components.
- **Guards**: Guards are used to protect certain routes in your application from unauthorized access. For example, you might have an authentication guard that requires users to log in before accessing certain pages.
- **Login and Register Models**: These are components that provide the user interface for logging in and registering for your application. They typically contain forms that collect user information and submit it to your backend for processing.
- **Navbar**: The navbar is a common component that appears at the top of your application and provides navigation links to other parts of your application.
- **Assets**: Assets are static files that are used by your application, such as images, fonts, and stylesheets. These files are typically stored in the "assets" folder in your project.

- **Environments**: Environments are used to define configuration settings for your application. You might have different environments for development, staging, and production, each with their own set of configuration settings.

These components work together to create a functional Angular application with a user-friendly interface and secure access controls.

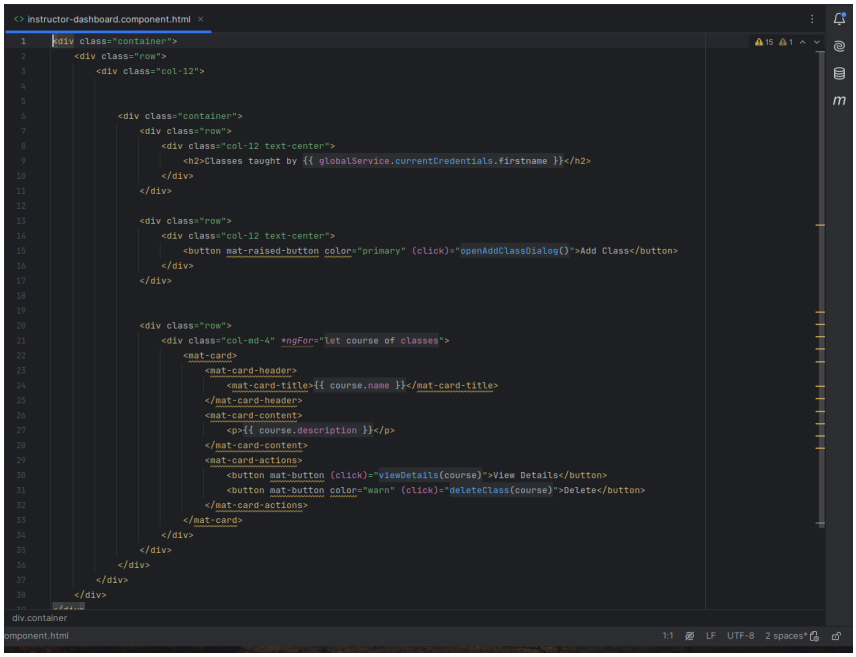



Figure 34: The instructor-dashboard.component.html file

In an Angular application, the HTML file is where you define the user interface for your application. The HTML file is typically associated with a specific component and contains the markup that defines the layout and content of that component.

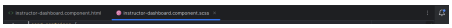


Figure 35: The instructor-dashboard.component.scss file

In an Angular application, the `.scss` file is where you define the styles for your application components using the Sass preprocessor syntax. The SCSS file is typically associated with a specific component and contains the styles that define the visual presentation of that component.

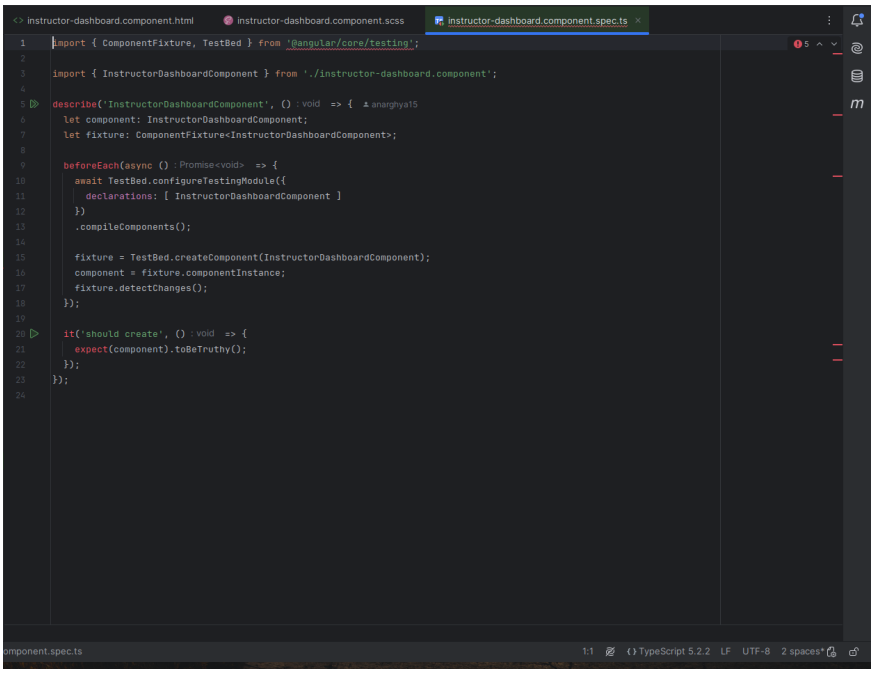



Figure 36: The instructor-dashboard.component.spec.ts file

In an Angular application, the `.spec.ts` file is where you write unit tests for your application components. The `.spec.ts` file is typically associated with a specific component and contains the test cases that ensure the component behaves correctly under different conditions.

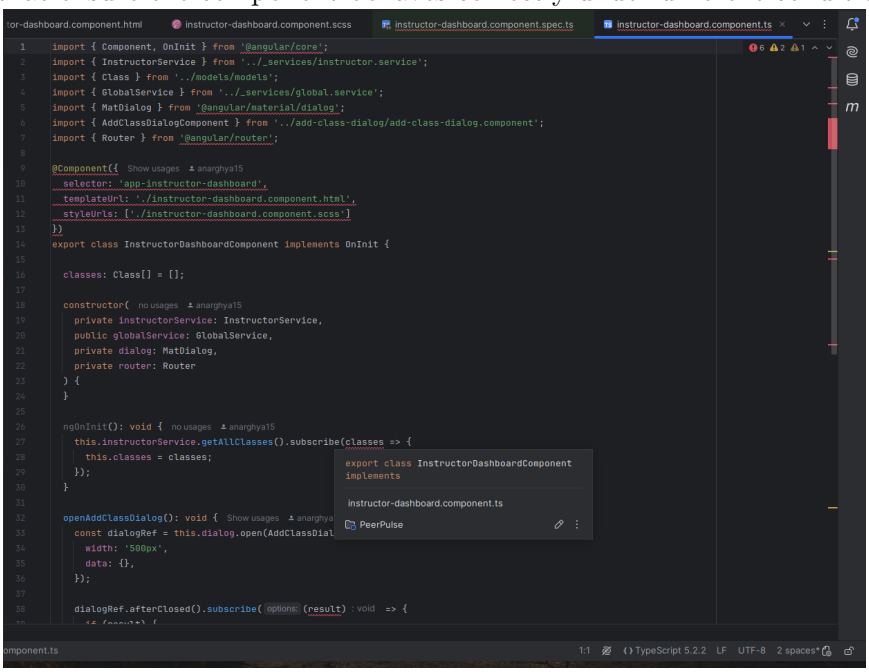



Figure 37: The instructor-dashboard.component.ts file

In an Angular application, the `.ts` file is where you define the behavior and functionality of your application components using TypeScript code. The `.ts` file is typically associated with a specific component and contains the class that defines the component's properties, methods, and lifecycle hooks.

## 12.2 nginx.conf

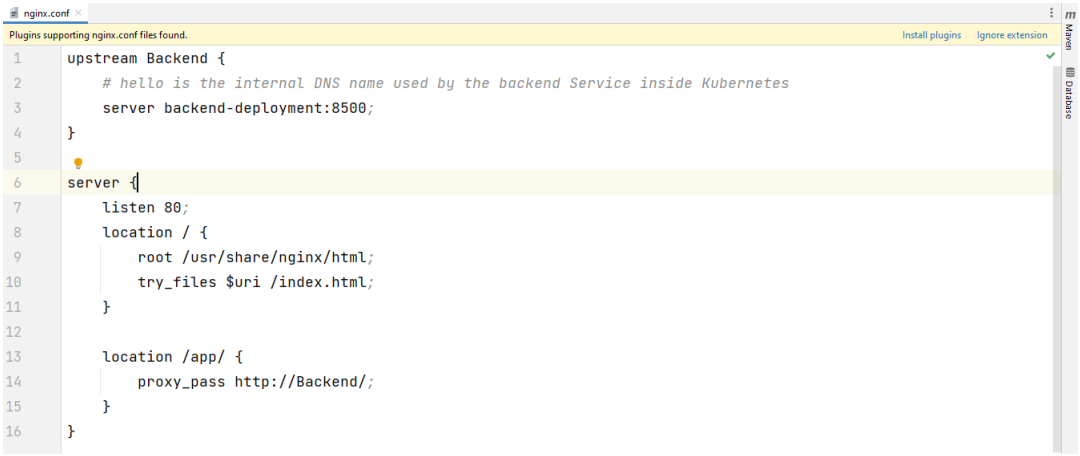



Figure 38: The `nginx.conf` file

This is an Nginx configuration file that defines a reverse proxy to route requests to a backend service running on Kubernetes.

The upstream block defines the backend service's internal DNS name as `backend-deployment` and port number 8500. This is the address that Nginx will use to route requests to the backend service.

The server block defines the listening port for incoming requests on port 80. The location block with the root directive specifies the root directory for serving static files when a request is made to the root path "/". The `try_files` directive attempts to find the requested file and serves `index.html` if the requested file is not found.

The second location block, with the path "/app/", is the one that proxies requests to the backend service. The `proxy_pass` directive forwards requests to the "Backend" upstream block that we defined earlier, which contains the backend service's internal DNS name and port number.

This configuration file sets up an Nginx reverse proxy to route requests to a backend service running on Kubernetes. Requests to the root path are served with static files, while requests to the "/app/" path are forwarded to the backend service.

The purpose of using a reverse proxy server like NGINX in front of a backend service is to provide an additional layer of security, scalability, and flexibility to the overall system. NGINX is being used as a reverse proxy to forward requests for the "/app/" location to a backend service running in a Kubernetes cluster. This allows for improved performance, scalability, and security of the backend service, while also allowing NGINX to handle tasks such as load balancing and SSL termination.

## 12.3 Backend

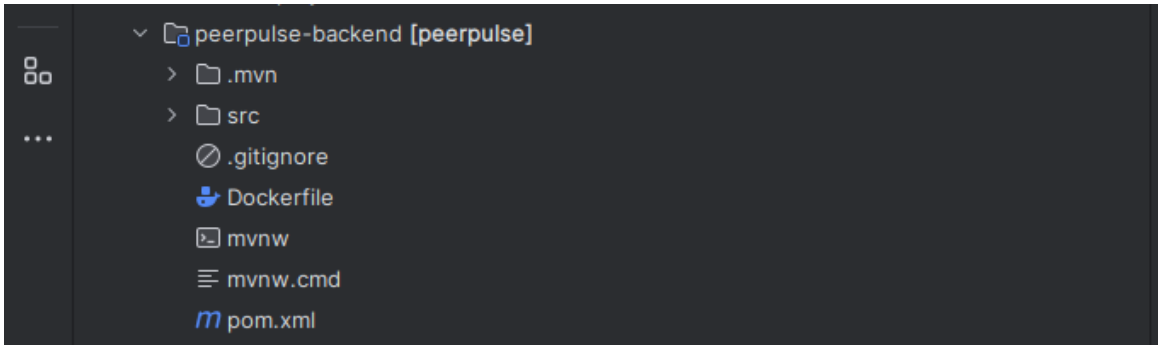



Figure 39: The directory structure of the backend

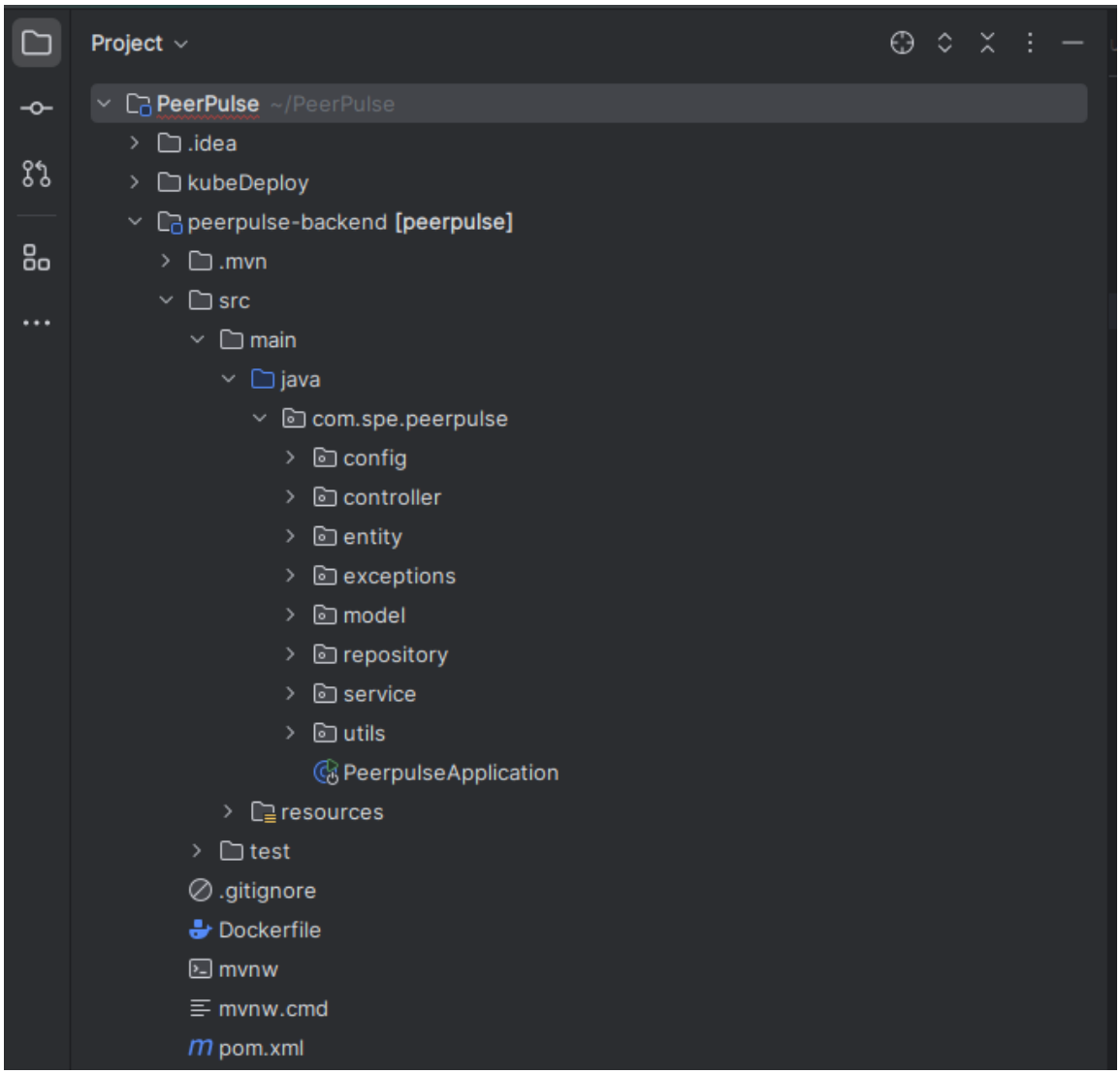



Figure 40: The extended directory structure of the backend

In this Spring Boot application, there are several common components that are encountered:

- **Config**: This component contains configuration files that define properties and settings for the application. Spring Boot provides a number of ways to configure an application, including using Java-based configuration, XML configuration, and properties files.

- **Controller**: This component defines the REST API endpoints for the application. Controllers receive incoming HTTP requests and return responses. They can also perform validation, authentication, and authorization.
- **Entity**: This component defines the data model for the application. Entities are typically mapped to database tables and represent the data that is stored or retrieved by the application.
- **Exceptions**: This component defines custom exceptions that can be thrown by the application. These exceptions can be used to provide more detailed error messages and handle specific error scenarios.
- **Model**: This component defines the data transfer objects (DTOs) used to transfer data between the client and the server. DTOs are often used to simplify the data model and provide a more concise representation of the data being transferred.
- **Repository**: This component defines the data access layer of the application. Repositories are used to interact with the database and provide CRUD (Create, Read, Update, Delete) operations on the entities.
- **Services**: This component contains the business logic of the application. Services are responsible for processing requests, manipulating data, and coordinating the interactions between the different components of the application.
- **Utils**: This component contains utility classes and methods that are used throughout the application. This can include helper classes for parsing data, formatting dates, or performing other common tasks.

Using these components, we can create a well-structured and maintainable codebase that is easy to understand and extend.

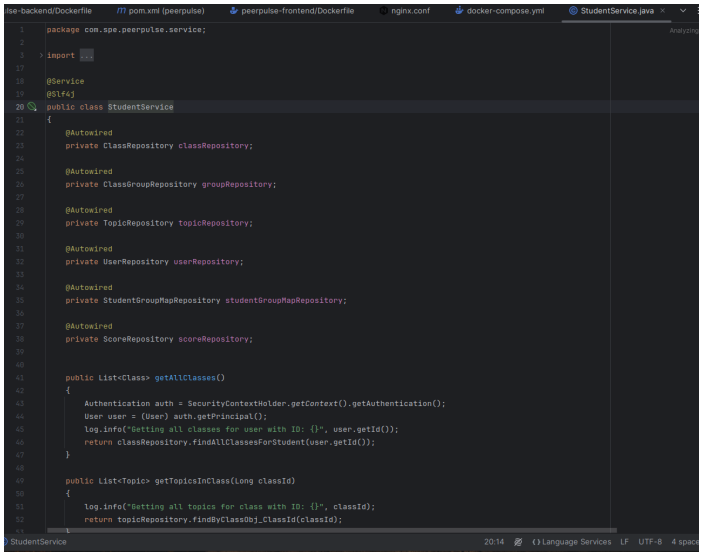



Figure 41: A snippet of the `StudentService.java` file

### 12.3.1 Login through JWT

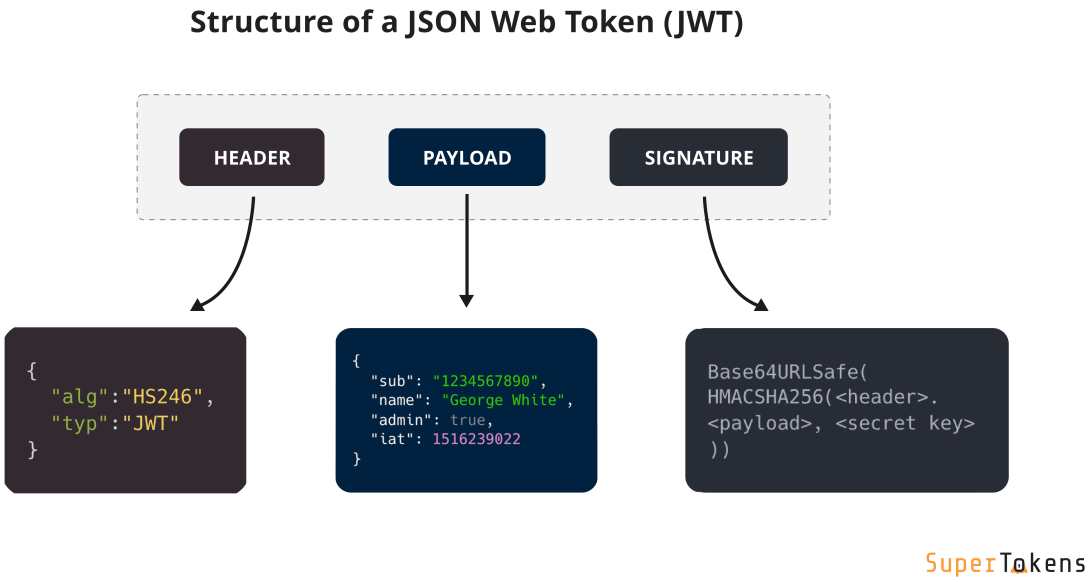
JSON Web Token (JWT) is a standard for securely transmitting information between parties as a JSON object. In a Spring Boot application, JWT is commonly used as a means of stateless authentication and authorization.

JWT consists of three parts: a header, a payload, and a signature. The header contains metadata about the token, such as the type of signature algorithm used. The payload contains

the claims, which are statements about an entity (typically the user) and additional data. The signature is used to verify that the message was not changed along the way and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is and was authorized to sign the token.

In a Spring Boot application, JWT can be used as a means of authentication and authorization by generating a token when a user logs in or authenticates, and then including that token with any subsequent requests made by the user. The server can then use the information in the token to determine if the user is authorized to access the requested resource.

Spring Security provides built-in support for JWT-based authentication and authorization in a Spring Boot application. This involves configuring a filter to intercept incoming requests, extracting the JWT token from the request, and validating the token. Once the token is validated, the user can be authorized to access the requested resource.




```java
@Service
@Slf4j
public class JwtService {
    private static final String SECRET_KEY = "67566B5970337336763979244226452848 2B4D6251655468576D5A7134743777";

    public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String getSecretKey() {
        return SECRET_KEY;
    }


    // Generate a jwt token in case a new user registers / logs without extra claims
    public String generateToken(UserDetails userDetails) { return generateToken(new HashMap<>(), userDetails); }

    // Generate a jwt token in case a new user registers / logs in containing extra claims
```

Figure 42: A snippet of the `JwtService.java` file

## 12.4 Database

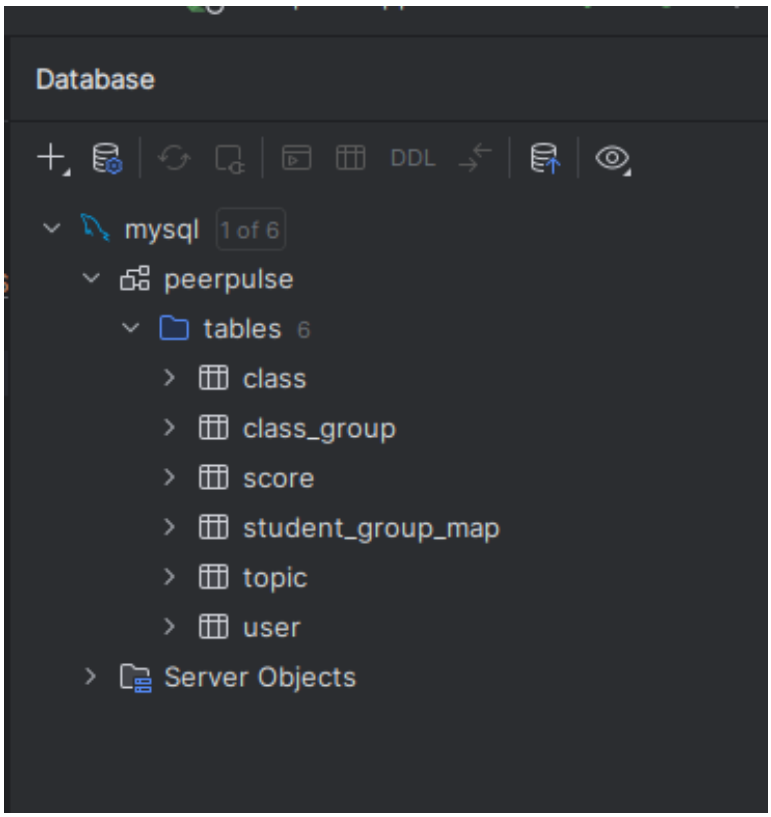
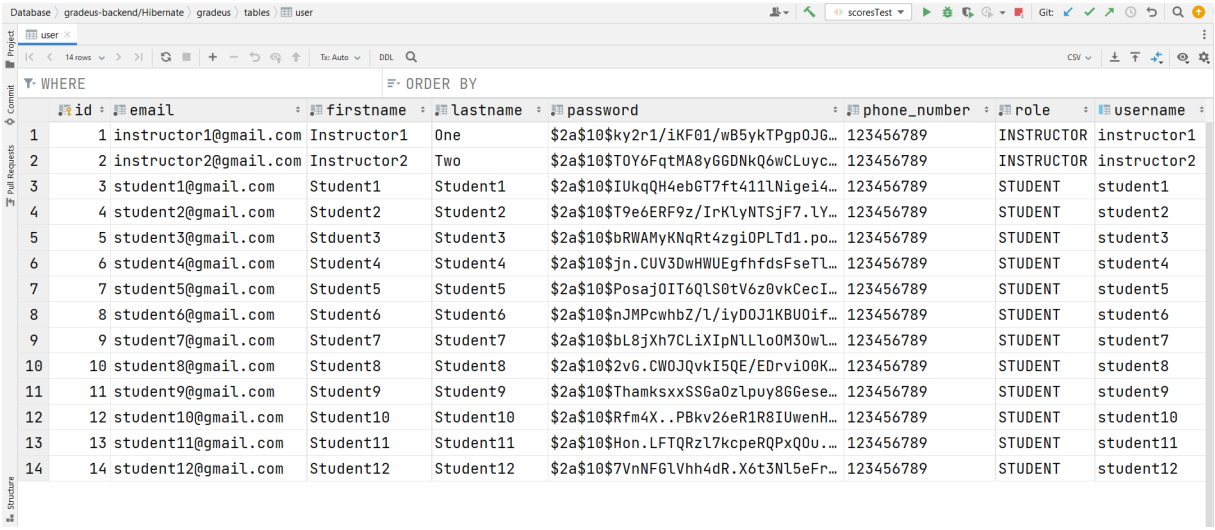



Figure 43: The schema and tables of the database



Database › gradeus-backend/Hibernate › gradeus › tables › user

user

WHERE ORDER BY

| | id | email | firstname | lastname | password | phone_number | role | username |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | instructor1@gmail.com | Instructor1 | One | $2a$10$ky2r1/iKF01/wB5ykTPgpOJG… | 123456789 | INSTRUCTOR | instructor1 |
| 2 | 2 | instructor2@gmail.com | Instructor2 | Two | $2a$10$TOY6FqtMA8yGGDNkQ6wCLuyc… | 123456789 | INSTRUCTOR | instructor2 |
| 3 | 3 | student1@gmail.com | Student1 | Student1 | $2a$10$IUkqQH4ebGT7ft411lNigei4… | 123456789 | STUDENT | student1 |
| 4 | 4 | student2@gmail.com | Student2 | Student2 | $2a$10$T9e6ERF9z/IrKlyNTSjF7.lY… | 123456789 | STUDENT | student2 |
| 5 | 5 | student3@gmail.com | Stduent3 | Student3 | $2a$10$bRWAMyKNqRt4zgiOPLTd1.po… | 123456789 | STUDENT | student3 |
| 6 | 6 | student4@gmail.com | Student4 | Student4 | $2a$10$jn.CUV3DwHWUEgfhfdsFseTl… | 123456789 | STUDENT | student4 |
| 7 | 7 | student5@gmail.com | Student5 | Student5 | $2a$10$PosajOIT6QlS0tV6z0vkCecI… | 123456789 | STUDENT | student5 |
| 8 | 8 | student6@gmail.com | Student6 | Student6 | $2a$10$nJMPcwhbZ/l/iyDOJ1KBUOif… | 123456789 | STUDENT | student6 |
| 9 | 9 | student7@gmail.com | Student7 | Student7 | $2a$10$bL8jXh7CLiXIpNlLloOM3Owl… | 123456789 | STUDENT | student7 |
| 10 | 10 | student8@gmail.com | Student8 | Student8 | $2a$10$2vG.CWOJQvkI5QE/EDrviO0K… | 123456789 | STUDENT | student8 |
| 11 | 11 | student9@gmail.com | Student9 | Student9 | $2a$10$ThamksxxSSGaOzlpuy8GGese… | 123456789 | STUDENT | student9 |
| 12 | 12 | student10@gmail.com | Student10 | Student10 | $2a$10$Rfm4X..PBkv26eR1R8IUwenH… | 123456789 | STUDENT | student10 |
| 13 | 13 | student11@gmail.com | Student11 | Student11 | $2a$10$Hon.LFTQRzl7kcpeRQPxQOu.… | 123456789 | STUDENT | student11 |
| 14 | 14 | student12@gmail.com | Student12 | Student12 | $2a$10$7VnNFGlVhh4dR.X6t3Nl5eFr… | 123456789 | STUDENT | student12 |

Figure 44: The user table of the database

### 12.4.1 ER Diagram

An ER (Entity-Relationship) diagram is a visual representation of the relationships between entities in a MySQL database. It is a tool used to design and model the structure of a database and its relationships.

In an ER diagram, entities are represented as boxes, and relationships between entities are represented as lines connecting the boxes. The boxes contain the attributes of the entity, which describe the properties of the entity.

There are several components of an ER diagram:

- **Entities**: An entity is a real-world object or concept that is represented in the database.

For example, in a database for a school, entities might include students, teachers, courses, and classrooms.

- **Attributes**: Attributes are the properties or characteristics of an entity. For example, the attributes of a student might include their name, ID number, and date of birth.
- **Relationships**: Relationships describe how entities are related to each other. For example, a student might be enrolled in one or more courses, and a course might have many students enrolled in it.
- **Cardinality**: Cardinality describes the number of instances of one entity that can be related to another entity. For example, a student can be enrolled in one or many courses, but a course can have many students enrolled in it.
- **Primary key**: A primary key is a unique identifier for each entity in the database. It is used to ensure that each entity can be uniquely identified and to enforce data integrity.

ER diagrams are useful for database designers and developers because they provide a clear and visual representation of the relationships between entities in a database. They can be used to communicate database designs to stakeholders and to ensure that the database structure is well-designed and efficient. Additionally, ER diagrams can be used to generate the SQL code needed to create the database schema and tables.

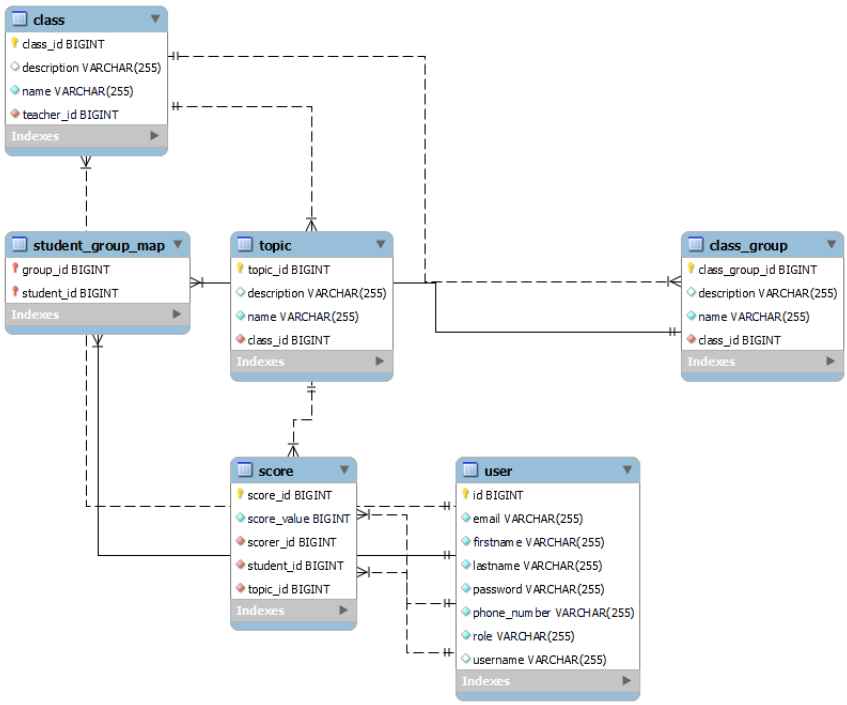



Figure 45: The ER diagram of the database design

## 12.5 Logging

Logging is an important aspect of any application, including Spring Boot applications. It allows you to track and troubleshoot issues that may arise during the application runtime.

Slf4j (Simple Logging Facade for Java) is a logging facade that provides a simple, unified interface for different logging frameworks. It is commonly used in Spring Boot applications to abstract away the underlying logging implementation and provide a consistent logging API across different environments.

```java
public List<Score> getScoresInTopic(Long topicId)
{
    try
    {
        return scoreRepository.findByTopic_TopicId(topicId);
    } catch (Exception e)
    {
        log.error("An error occurred while getting scores for topicId={}", topicId, e);
        throw new RuntimeException("Failed to get scores", e);
    }
}
```

Figure 46: An example of logging in a backend service

## 12.6 Build Automation using Maven

Build automation is the process of automating the tasks involved in building and deploying software applications. It involves the use of software tools and scripts to automate the process of compiling, testing, and packaging software, enabling developers to quickly and easily build and deploy their applications.

Build automation is important for several reasons. First, it helps to improve the speed and efficiency of the software development process. By automating the build process, developers can quickly and easily build and deploy their applications, reducing the time and effort required to test and deploy new features and updates.

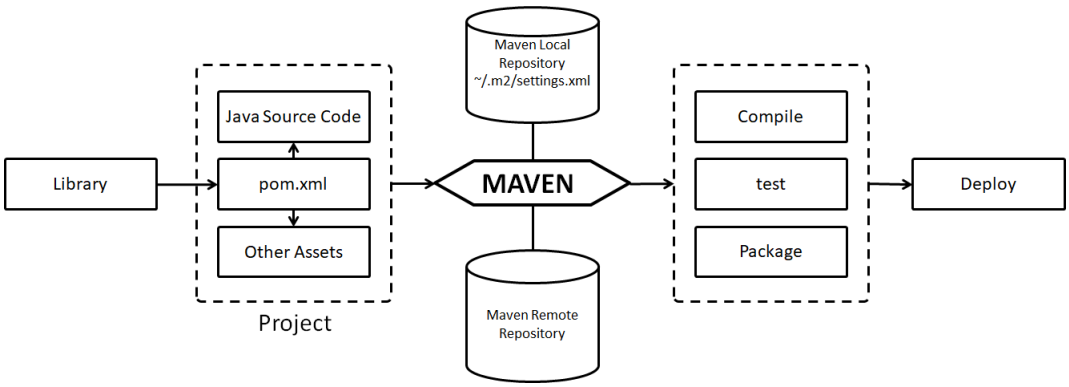



Figure 47: The architecture of Maven

Second, build automation helps to improve the quality of software applications. By automating the build process, developers can ensure that their applications are built and tested consistently, reducing the risk of errors and bugs that can occur when builds are done manually.

Third, build automation helps to reduce the risk of configuration errors and other issues that can arise when applications are deployed to different environments. By automating the build process, developers can ensure that their applications are configured correctly and consistently across different environments, reducing the risk of errors and bugs that can occur when applications are deployed manually.

Build automation typically involves the use of a build automation tool, such as Jenkins, Travis CI, or CircleCI. These tools provide a range of features and capabilities for automating the build process, including compiling code, running tests, and packaging applications.

Build automation tools typically use a build script, which is a set of instructions that tell the tool how to build and deploy the application. The build script can be written in a variety of languages, including Bash, Python, and Groovy.

Build automation also often involves the use of version control systems, such as Git or SVN.

Version control systems allow developers to manage changes to their code over time, enabling them to track changes, collaborate with other developers, and roll back changes if necessary.

Overall, build automation is a critical component of modern software development, enabling developers to quickly and easily build and deploy their applications. By automating the build process, developers can improve the speed and efficiency of the development process, improve the quality of their applications, and reduce the risk of errors and bugs that can occur when builds are done manually.

Maven is a popular build automation tool that is used to manage the build process for Java-based software projects. It is an open-source project that is maintained by the Apache Software Foundation and is widely used by developers and organizations around the world.

Maven provides a range of features and capabilities for automating the build process, including compiling code, running tests, and packaging applications. It also provides a powerful dependency management system, which allows developers to manage dependencies on external libraries and frameworks.

One of the key benefits of using Maven is its ability to manage the entire build process from a single, centralized configuration file. This configuration file, known as the POM (Project Object Model), defines the structure of the project, the dependencies, the build process, and other important information.

The POM file contains a range of information about the project, including the project name, version, and description, as well as information about the project's dependencies and plugins. The POM file can also be used to define the build process, including the compilation and packaging of the application.

Maven uses a declarative syntax, which means that developers can specify what they want Maven to do, rather than how to do it. This allows developers to focus on the high-level aspects of the build process, rather than getting bogged down in the details of how the build process is implemented.

Maven also provides a powerful dependency management system, which allows developers to manage dependencies on external libraries and frameworks. The dependency management system uses a centralized repository, which contains a range of libraries and frameworks that can be used in Maven projects.

Developers can specify the dependencies for their project in the POM file, and Maven will automatically download and manage the required dependencies. This makes it easy to manage dependencies across multiple projects, and helps to ensure that the correct versions of dependencies are used throughout the project.

Maven also provides a range of plugins, which can be used to extend the functionality of the build process. These plugins can be used for a variety of tasks, including running tests, generating documentation, and deploying the application to a production environment.

Maven is a powerful and flexible build automation tool that is widely used by Java developers around the world. Its ability to manage the entire build process from a single configuration file, its powerful dependency management system, and its range of plugins make it an indispensable tool for modern software development.

### 12.6.1 Installation

Sure, here are the steps to install Maven on Ubuntu:

1. Open a terminal window on your Ubuntu machine.
2. Update the package index of your system by running the following command:

   ```
   sudo apt update
   ```

3. Install the default JDK by running the following command: `sudo apt install default-jdk`

4. Once the installation is complete, verify that Java is installed on your system by running the following command: `java -version`

5. Next, download the latest version of Maven from the Apache Maven website. You can use the following command to download the latest version:

   `wget https://apache.osuosl.org/maven/maven-3/3.8.4/binaries/`

   `apache-maven-3.8.4-bin.tar.gz` (Note: replace the version number with the latest version available on the Apache Maven website)

6. Once the download is complete, extract the downloaded archive to the /opt directory using the following command: `sudo tar -xf apache-maven-3.8.4-bin.tar.gz -C /opt` (Note: replace the version number with the version you downloaded)

7. Next, create a symbolic link to the Maven installation directory using the following command: `sudo ln -s /opt/apache-maven-3.8.4 /opt/maven`

   (Note: replace the version number with the version you downloaded)

8. Now, open the /etc/profile file using a text editor of your choice:

   `sudo nano /etc/profile`

   Add the following lines to the end of the file:

   ```
   export M2_HOME=/opt/maven
   export MAVEN_HOME=/opt/maven
   export PATH=${M2_HOME}/bin:${PATH}
   ```

   Save the file and exit the text editor.

9. To apply the changes, reload the system-wide environment variables using the following command:

   `source /etc/profile`

10. Verify that Maven is installed on your system by running the following command: `mvn -version`

    If everything was installed correctly, you should see the version of Maven installed on your system.

### 12.6.2 Basic Commands

Maven is a powerful build automation tool that provides a wide range of features and capabilities. Here are some basic Maven commands that you can use to start building and managing your Java-based projects:

- **`mvn clean`**: This command cleans up the project by deleting the target directory and any other generated files. This is useful when you want to start fresh with a new build.
- **`mvn compile`**: This command compiles the Java source code in the project. The compiled code is stored in the target/classes directory.
- **`mvn package`**: This command packages the compiled code into a distributable format, such as a JAR or WAR file. The packaged file is stored in the target directory.
- **`mvn install`**: This command installs the packaged artifact into the local repository, making it available for use by other projects on the same machine.

- **`mvn test`**: This command runs the unit tests associated with the project. The tests are run using the JUnit testing framework, and the results are displayed in the console.
- **`mvn dependency:tree`**: This command displays the dependency tree for the project, showing all of the dependencies that the project depends on, as well as their transitive dependencies.
- **`mvn help`**: This command displays help information about Maven, including a list of available commands and their syntax.

### 12.6.3 POM File

The POM (Project Object Model) file is a critical component of the Maven build automation tool. It is an XML file that contains information about the project, its dependencies, and its build process. The POM file is used by Maven to manage the build process for the project, and it provides a centralized location for defining the structure and configuration of the project.

The POM file contains a range of information about the project, including its name, version, and description. It also contains information about the project's dependencies, which are managed by Maven's dependency management system. The POM file can also be used to specify the build process for the project, including the compilation and packaging of the application.

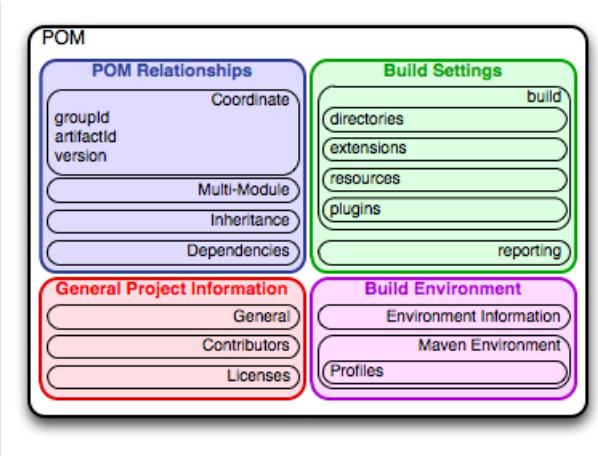



Figure 48: Structure of the `pom.xml` file

One of the key benefits of the POM is that it provides a centralized location for managing the configuration of the project. This makes it easy to maintain a consistent structure and configuration across the project, and helps to ensure that the project is built and packaged correctly.

The POM file is typically located in the root directory of the project, and it is named "pom.xml". The file is structured using XML, and it is divided into several sections, each of which corresponds to a different aspect of the project's configuration.

The sections of the POM file include:

- **Project**: This section contains information about the project, including its name, version, description, and packaging type (e.g. JAR, WAR, or EAR).
- **Properties**: This section contains a set of properties that can be used throughout the POM file. These properties can be used to define values that are used multiple times throughout the POM file, such as the version of a library or the path to a file.

- **Dependencies**: This section contains information about the project's dependencies. It specifies the libraries and frameworks that the project depends on, and it includes information about the version and scope of each dependency.
- **Build**: This section specifies the build process for the project. It includes information about the plugins and goals that are used to build and package the application.
- **Profiles**: This section allows developers to define different configurations for the project based on different environments, such as development, testing, and production.

The Project section of the POM file is perhaps the most important. It defines the basic information about the project, such as its name, version, and description. It also specifies the packaging type for the project, which determines the type of artifact that is produced by the build process.

The Properties section of the POM file allows developers to define variables that can be used throughout the POM file. These variables can be used to define values that are used multiple times throughout the POM file, making it easier to maintain a consistent structure and configuration across the project.

The Dependencies section of the POM file is used to manage the project's dependencies. It specifies the libraries and frameworks that the project depends on, and it includes information about the version and scope of each dependency. Maven's dependency management system uses this information to manage the project's dependencies, ensuring that the correct versions of dependencies are used throughout the project.

The Build section of the POM file specifies the build process for the project. It includes information about the plugins and goals that are used to build and package the application. Maven provides a wide range of plugins that can be used to extend the functionality of the build process, such as running tests, generating documentation, and deploying the application to a production environment.

The Profiles section of the POM file allows developers to define different configurations for the project based on different environments, such as development, testing, and production. Each profile can define its own set of dependencies, build process, and other configuration options, allowing developers to tailor the project to specific environments.

The POM file is a critical component of the Maven build automation tool. It provides a centralized location for managing the configuration of the project, and it helps to ensure that the project is built and packaged correctly. By using the POM, developers can create consistent, well-structured projects that are easy to build and maintain.

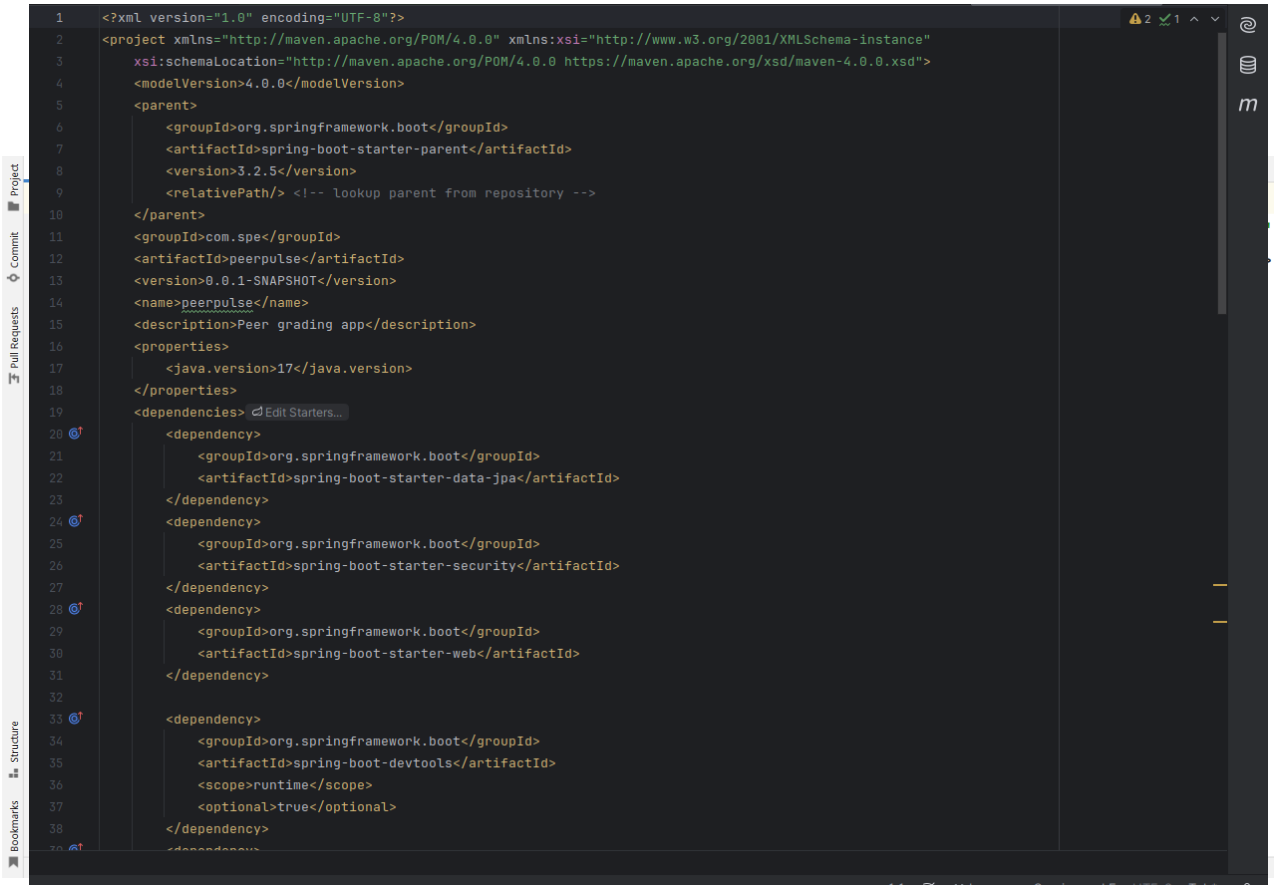



Figure 49: A snippet of the `pom.xml` file for the application

# 13 Continous Monitoring

Continuous monitoring is an important component of DevOps, as it helps teams to ensure that their systems and applications are functioning as expected and are secure. In the context of DevOps, continuous monitoring involves the use of automated tools and processes to monitor systems, applications, and infrastructure in real-time, allowing teams to quickly detect and respond to potential issues.

Continuous monitoring in DevOps involves several key steps. First, teams need to establish a baseline for expected behavior for their systems and applications. This may involve the use of monitoring tools and analytics to identify patterns and trends in system behavior. Once a baseline has been established, teams can use continuous monitoring tools to identify any deviations from expected behavior, allowing them to quickly respond and address any potential issues.

Continuous monitoring in DevOps typically involves the use of tools and techniques such as log analysis, performance monitoring, and security monitoring. Log analysis involves analyzing the logs generated by systems and applications, looking for patterns and anomalies that may indicate potential issues. Performance monitoring involves monitoring systems and applications for issues such as slow response times or resource contention, allowing teams to quickly identify and address any bottlenecks or performance issues. Security monitoring involves monitoring systems and applications for potential security issues, such as unauthorized access or data breaches.

Continuous monitoring in DevOps is important for several reasons. First, it allows teams to quickly detect and respond to potential issues, reducing the risk of downtime or service disruptions. Second, it helps teams to identify potential performance issues and bottlenecks, allowing them to proactively address these issues before they become a problem. Finally, it helps teams to ensure that their systems and applications are secure, reducing the risk of data breaches and other security incidents.

To implement continuous monitoring in DevOps, teams typically need to establish clear policies and procedures for monitoring and responding to potential issues. This may involve the use of automated tools and processes, such as automated incident response tools and workflows. Teams also need to establish clear roles and responsibilities for monitoring and responding to potential issues, ensuring that everyone on the team understands their role in maintaining the health and security of the systems and applications they are responsible for.

Continuous monitoring is an important component of DevOps, as it helps teams to ensure that their systems and applications are functioning properly and are secure. By using automated tools and processes to monitor systems and applications in real-time, teams can quickly detect and respond to potential issues, reducing the risk of downtime and service disruptions, and ensuring that their systems and applications are functioning as expected.

## 13.1 The ELK Stack

The ELK stack is a collection of three open-source tools that are commonly used for log management and analysis. The three components of the ELK stack are Elasticsearch, Logstash, and Kibana, and together they provide a powerful platform for collecting, analyzing, and visualizing log data.

**Elasticsearch** is a distributed search and analytics engine that is used to store and search large volumes of data. It is designed to be highly scalable and can handle large volumes of data in real-time. Elasticsearch is used to store and index log data, which can then be searched and analyzed using the other components of the ELK stack.

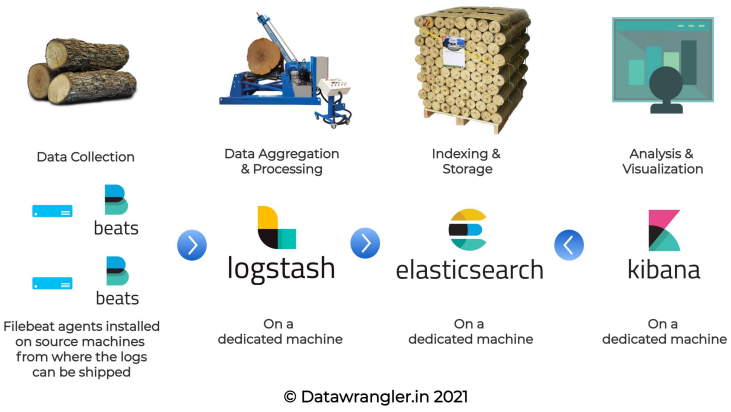



Figure 50: Architecture of the ELK Stack

**Logstash** is a data processing tool that is used to collect, transform, and enrich log data. It is designed to collect log data from a variety of sources, including log files, databases, and message queues. Logstash can parse and transform log data, enrich it with additional information, and then send it to Elasticsearch for storage and analysis.

**Kibana** is a data visualization and exploration tool that is used to analyze and visualize log data. It provides a range of powerful visualization tools, including line charts, bar charts, scatter plots, and heatmaps, that can be used to analyze log data in real-time. Kibana provides a range of filters and query tools, allowing users to search and filter log data based on specific criteria.

Together, the three components of the ELK stack provide a powerful platform for log management and analysis. Log data is collected and processed using Logstash, and then stored and indexed in Elasticsearch. Kibana is used to visualize and explore the log data, allowing users to identify patterns, trends, and anomalies in the data.

The ELK stack is used by a wide range of organizations for log management and analysis. It is particularly useful for organizations that need to analyze and monitor large volumes of log data, such as web applications, IT infrastructure, and security systems.

One of the key benefits of the ELK stack is its flexibility and extensibility. It is designed to be highly customizable, and users can easily extend its functionality using plugins and APIs. This allows organizations to tailor the ELK stack to their specific needs, and to integrate it with other tools and platforms.

The ELK stack is a powerful and flexible platform for log management and analysis. Its three components, Elasticsearch, Logstash, and Kibana, provide a range of powerful tools for collecting, analyzing, and visualizing log data, making it an ideal choice for organizations that need to monitor and analyze large volumes of log data in real-time.

### 13.1.1 Filebeat

Filebeat is a lightweight open-source data shipper that is used to collect, parse, and forward log data from different sources to various destinations, such as Elasticsearch, Logstash, and Kafka. It is part of the Elastic Stack, which is a suite of tools for managing and analyzing data.

Filebeat is designed to be simple and efficient, making it easy to deploy and use. It can read and tail log files, parse different log formats, and send the data to one or more destinations

in near real-time. Filebeat supports different input types, such as files, syslog, and TCP/UDP sockets.

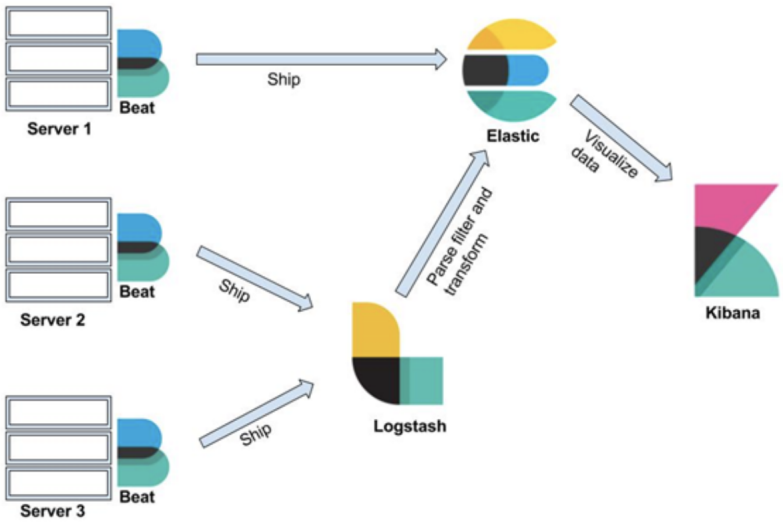



Figure 51: An illustration of using Filebeat

Filebeat is often used in combination with other Elastic Stack components to create a complete log management and analysis solution. For example, Filebeat can be used to collect log data from different sources, such as servers and applications, and send it to Logstash for further processing and analysis. Alternatively, Filebeat can send the data directly to Elasticsearch, where it can be indexed and searched using Kibana.

Filebeat is a powerful and flexible tool for collecting and forwarding log data, making it an essential component of many modern log management and analysis systems.

## 13.2 Elastic Cloud

Elastic Cloud is a cloud-based service that provides a fully managed and scalable version of the Elastic Stack, a set of powerful and scalable tools for data analysis and search. Elastic Cloud is designed to simplify the process of deploying and managing the Elastic Stack, providing a range of features and benefits for businesses and organizations of all sizes.


One of the key features of Elastic Cloud is its ability to provide a fully managed and scalable version of the Elastic Stack. This means that businesses and organizations can easily deploy and manage the Elastic Stack without having to worry about the underlying infrastructure or

the complexities of managing and scaling the software. This allows organizations to focus on their core business goals and objectives, rather than spending time and resources on managing infrastructure and software.

Elastic Cloud provides a range of powerful features for data analysis and search. This includes Elasticsearch, a powerful search and analytics engine that is designed to handle large volumes of data and provide fast and accurate search results. It also includes Kibana, a data visualization and exploration tool that is used to analyze and visualize large datasets. Kibana provides a range of powerful visualization tools, including line charts, bar charts, scatter plots, and heatmaps.

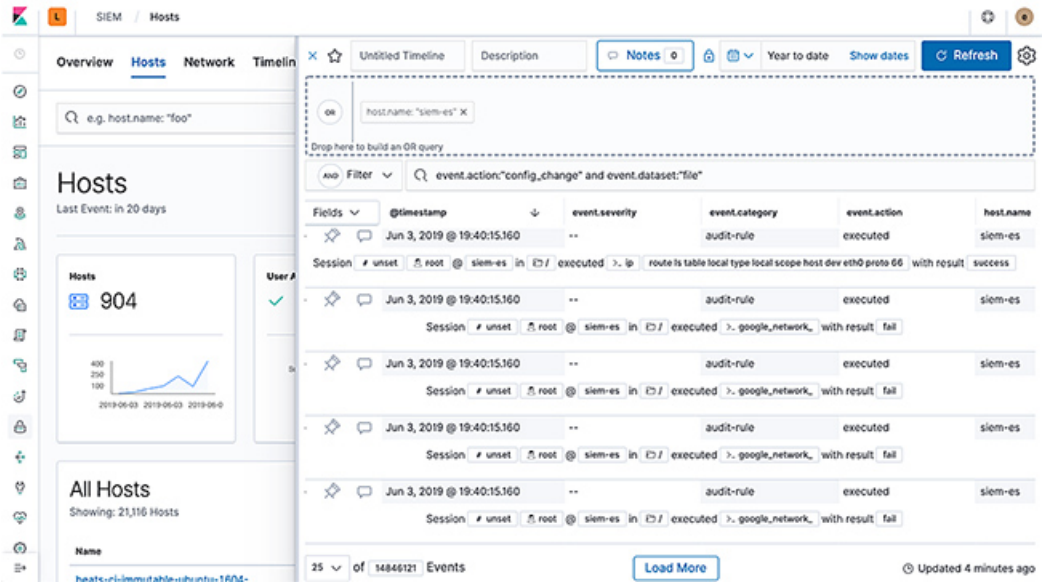



Figure 52: A sample Elastic Cloud dashboard

Elastic Cloud also provides a range of features for security and compliance. This includes role-based access control, which allows organizations to control access to data and features based on specific user roles and permissions. It also includes encryption at rest and in transit, which ensures that data is protected from unauthorized access and theft.

Elastic Cloud is designed to be highly scalable and flexible, allowing organizations to easily scale up or down based on their changing needs. It provides a range of deployment options, including dedicated, shared, and multi-tenant environments, allowing organizations to choose the deployment option that best fits their needs.

Elastic Cloud also provides a range of integration options, allowing organizations to easily integrate the Elastic Stack with other tools and platforms. This includes integrations with popular cloud platforms such as AWS and Azure, as well as a range of plugins and APIs that allow organizations to extend the functionality of the Elastic Stack.

Elastic Cloud is a powerful and flexible cloud-based service that provides a fully managed and scalable version of the Elastic Stack. Its powerful features for data analysis and search, along with its security and compliance features, make it a valuable tool for businesses and organizations of all sizes. Its scalability and flexibility, along with its integration options, make it a powerful tool for organizations that need to scale up or down quickly based on their changing needs.

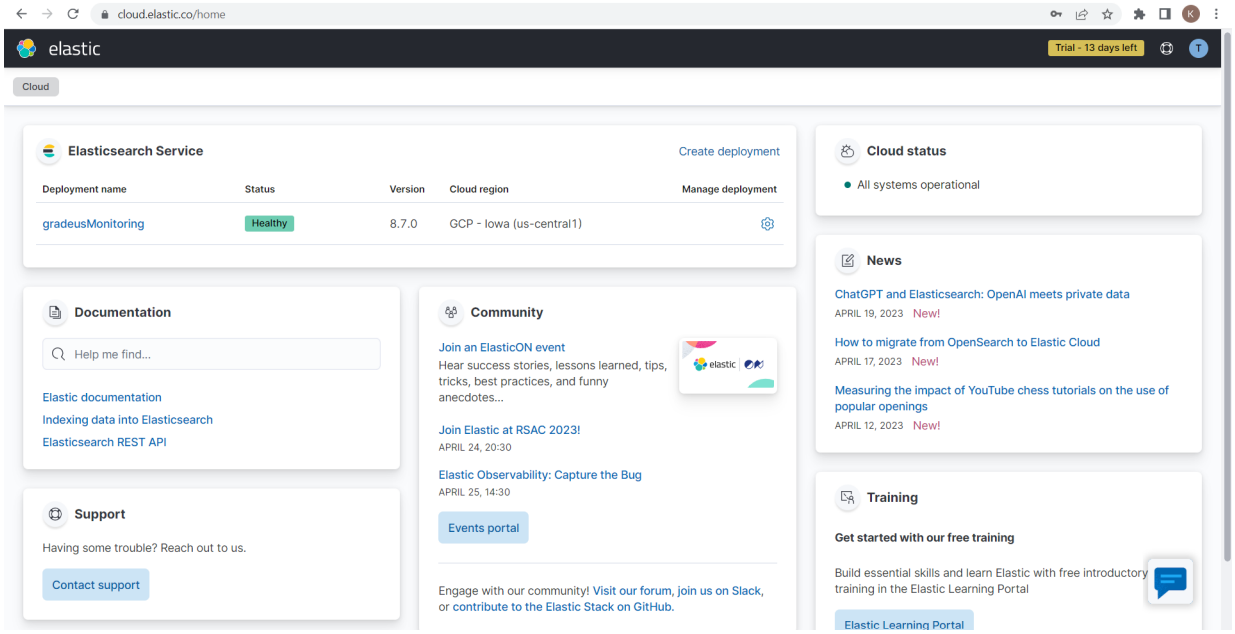



Figure 53: The Elastic dashboard of our application

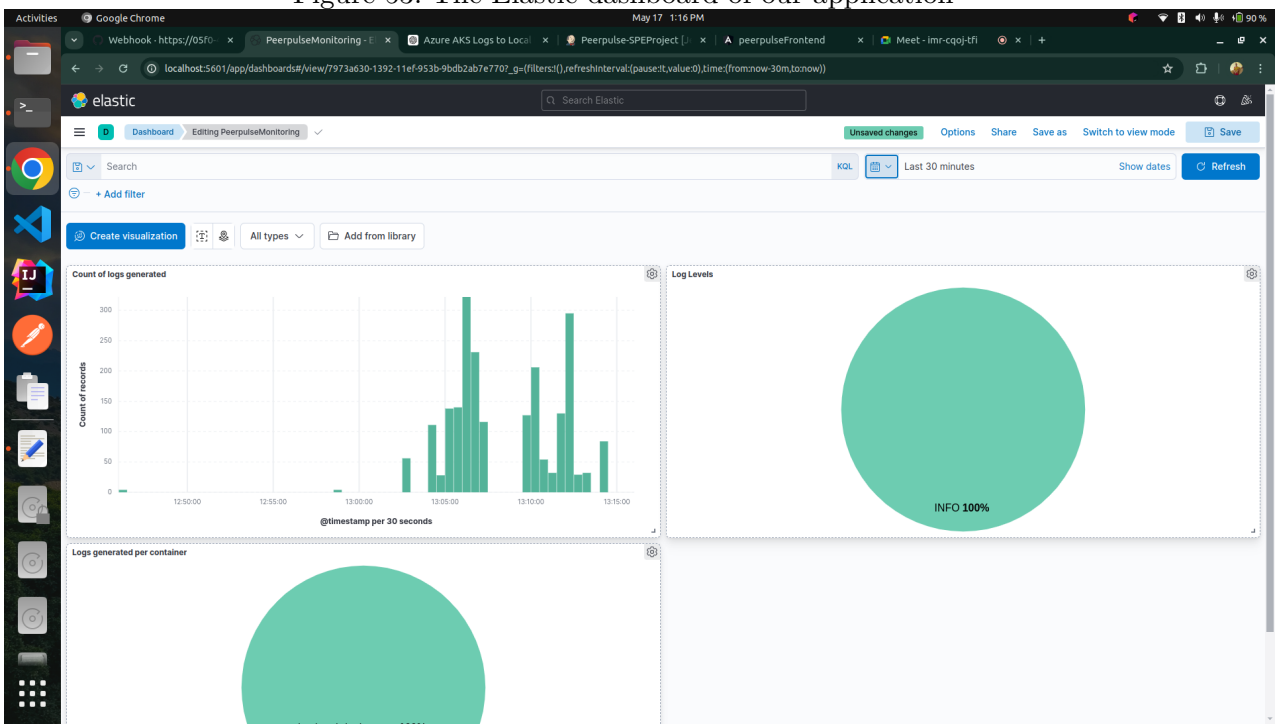



Figure 54: The monitoring logs of the application

### 13.2.1 Account Creation

Creating an account on Elastic Cloud is a simple and straightforward process that can be completed in just a few steps. Here's a step-by-step guide on how to create an account on Elastic Cloud:

- Go to the Elastic Cloud website:

  The first step is to go to the Elastic Cloud website at `https://www.elastic.co/cloud/`. From there, click on the "Sign Up" button in the top right corner of the page.

- Provide your email address:

On the next page, you will be prompted to provide your email address. Enter your email address and click on the "Continue" button.

- Choose a plan:

  On the next page, you will be asked to choose a plan. Elastic Cloud offers a range of plans, including Free, Basic, Standard, Gold, Platinum, and Enterprise. Choose the plan that best fits your needs and click on the "Continue" button.

- Verify your email address:

  You will receive an email from Elastic Cloud with a verification link. Click on the link to verify your email address.

- Set up your account:

  Once your email address is verified, you will be prompted to set up your account. This includes providing your name, company name, and password. Choose a strong password that meets the minimum security requirements, and click on the "Create Account" button.

- Choose a deployment:

  After creating your account, you will be prompted to choose a deployment. Elastic Cloud offers a range of deployment options, including Elasticsearch, Kibana, APM, and more. Choose the deployment option that best fits your needs and click on the "Create Deployment" button.

- Configure your deployment:

  Once you have chosen your deployment, you will be prompted to configure it. This includes choosing the region, setting up security options, and configuring any additional features or settings. Follow the prompts to configure your deployment, and click on the "Create" button when you are finished.

- Start using Elastic Cloud:

  Once your deployment is created, you can start using Elastic Cloud to analyze and visualize your data. You can access your deployment from the Elastic Cloud dashboard, which provides a range of features and tools for managing and monitoring your deployment.

Creating an account on Elastic Cloud is a simple and straightforward process that can be completed in just a few steps. By following these steps, you can quickly and easily set up your account and start using the powerful features and tools of Elastic Cloud to analyze and visualize your data.

## 13.3 Logstash

Logstash is an open-source data processing tool that is used to collect, transform, and enrich data from a range of sources. It is part of the Elastic Stack, a set of powerful and scalable tools for data analysis and search.

Logstash is designed to work with large volumes of data, making it ideal for businesses and organizations that need to process and analyze large volumes of data in real-time. It provides a range of powerful features for data processing and transformation, including the ability to collect data from a range of sources, parse and transform data, and enrich data with additional information.

One of the key features of Logstash is its ability to collect and process data from a wide range of sources, including logs, metrics, and other data sources. It provides a range of input plugins, allowing users to collect data from sources such as log files, databases, and message queues. It also provides a range of output plugins, allowing users to send data to a range of destinations, including Elasticsearch, Kafka, and various databases.

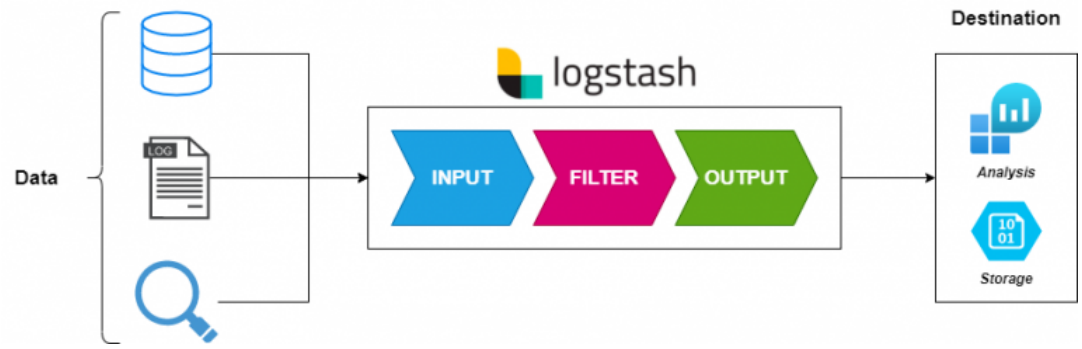



Figure 55: The architecture of Logstash

Logstash provides a range of powerful transformation capabilities, allowing users to parse, filter, and manipulate data as it flows through the pipeline. This includes the ability to extract fields from unstructured data, convert data types, and apply conditional logic based on specific criteria. Logstash also provides a range of filter plugins, allowing users to perform advanced data processing tasks such as geolocation, IP address lookup, and data masking.

Logstash provides a range of enrichment capabilities, allowing users to add additional information to data as it flows through the pipeline. This includes the ability to enrich data with information from external sources, such as GeoIP databases, and the ability to add metadata to data based on specific criteria.

Logstash is designed to be highly customizable and extensible, making it easy to integrate with other tools and platforms. It provides a range of APIs and plugins, allowing users to extend its functionality and integrate it with other data processing and analytics tools.

Logstash is a powerful and flexible data processing tool that is ideal for businesses and organizations that need to collect, transform, and enrich large volumes of data in real-time. Its powerful transformation and enrichment capabilities, along with its flexible architecture, make it a valuable tool for data processing and analysis in a wide range of industries and applications.

## 13.4 Kibana

Kibana is an open-source data visualization and exploration tool that is used to analyze and visualize large datasets. It is part of the Elastic Stack, a set of powerful and scalable tools for data analysis and search.

Kibana is designed to work with large volumes of data, making it ideal for businesses and organizations that need to analyze and visualize large datasets in real-time. It provides a range of powerful features for data exploration, including interactive visualizations, dynamic dashboards, and advanced search capabilities.

One of the key features of Kibana is its ability to connect to a range of data sources, including Elasticsearch, Logstash, and Beats. This allows users to easily explore and visualize data from a variety of sources, making it easier to identify patterns, trends, and anomalies in large datasets.

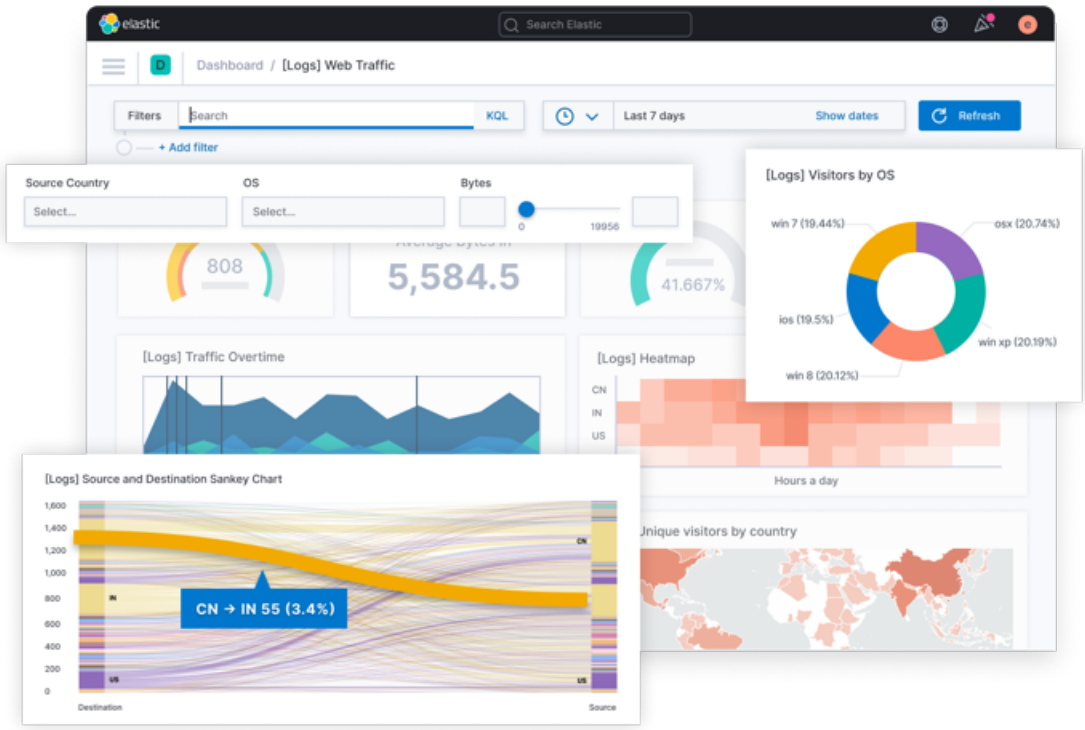



Figure 56: A sample Kibana dashboard

Kibana provides a range of powerful visualization tools, including line charts, bar charts, scatter plots, and heatmaps. These visualizations can be customized and arranged in dashboards, allowing users to monitor and analyze data in real-time. Kibana also provides a range of filters and query tools, allowing users to search and filter data based on specific criteria.

Kibana also provides a range of features for data analysis and monitoring. For example, it can be used to monitor website traffic in real-time, track user behavior, and identify trends and patterns. It can also be used to monitor system logs, network traffic, and other security-related data, making it a powerful tool for security and threat detection.

Kibana is designed to be highly customizable and extensible, making it easy to integrate with other tools and platforms. It provides a range of APIs and plugins, allowing users to extend its functionality and integrate it with other data analysis and visualization tools.

Kibana is a powerful and flexible data visualization and exploration tool that is ideal for businesses and organizations that need to analyze and visualize large datasets in real-time. Its powerful visualization tools, advanced search capabilities, and flexible architecture make it a valuable tool for data analysis and monitoring in a wide range of industries and applications.

# 14 Jenkins Pipeline Script

Jenkins Pipeline is a powerful tool for managing continuous integration and delivery (CI/CD) workflows. A Jenkins Pipeline script is a set of instructions written in a domain-specific language (DSL) that defines the entire automation process for building, testing, and deploying applications.

Pipeline scripts are defined as code that can be stored in a version control system, allowing for collaboration and versioning. The Jenkins Pipeline DSL provides a rich set of functions and plugins that enable users to define complex workflows, including parallel execution, conditional logic, and error handling.

Pipeline scripts can be defined in two ways: Declarative Pipeline and Scripted Pipeline. Declarative Pipeline syntax provides a straightforward way to define pipelines, while Scripted Pipeline syntax is more flexible and allows for more complex workflows.

Jenkins Pipeline scripts are a powerful tool for automating complex CI/CD workflows and providing greater visibility and control over the entire software delivery process.

A Jenkins pipeline script typically consists of several parts that define the stages and steps of a CI/CD workflow. Here are some common parts of a Jenkins pipeline script:

- **Pipeline definition**: This is where you define the pipeline using either Declarative or Scripted syntax. It includes information such as the agent or node where the pipeline will be executed, the stages that make up the pipeline, and the options for how the pipeline will run.
- **Environment variables**: You can define environment variables that will be used throughout the pipeline. These variables can be used to store information such as credentials, build numbers, or deployment targets.
- **Stages**: Stages represent the different phases of the pipeline, such as building, testing, and deploying. Each stage can include multiple steps, and the pipeline will only move to the next stage if the previous stage succeeds.
- **Steps**: Steps define the individual tasks that need to be executed within a stage. These can include shell scripts, commands to run tests, or steps to deploy the application.
- **Post-build actions**: After the pipeline is executed, you can define post-build actions, such as sending notifications or archiving artifacts.
- **Error handling**: You can define how the pipeline should handle errors, such as retrying failed steps, aborting the pipeline if a certain condition is met, or sending notifications when errors occur.

These parts work together to define the entire automation process for building, testing, and deploying applications in a CI/CD workflow using Jenkins Pipeline.

```
pipeline {
    agent any
    environment {
        frontendRepositoryName = "anarghya15/peerpulse-frontend"
        backendRepositoryName = "anarghya15/peerpulse-backend"
        KUBECONFIG = "/var/lib/jenkins/.kube/config"
        AZURE_SUBSCRIPTION_ID = credentials('azure-subscription-id')
        AZURE_TENANT_ID = credentials('azure-tenant-id')
        AZURE_CLIENT_ID = credentials('azure-service-principal')
        AZURE_CLIENT_SECRET = credentials('azure-client-secret')
        AZURE_RESOURCE_GROUP = "spe-project-peerpulse"
        AZURE_AKS_CLUSTER_NAME = "peerpulse-kubecluster"
        tag = "latest"
        frontendImage = ""
        backendImage = ""
    }
```

The `frontendRepositoryName` and `backendRepositoryName` variables specify the names of the Git repositories used for the frontend and backend code. The tag variable specifies the tag that will be applied to the Docker images that are built. The `frontendImage` and `backendImage` variables are initially set to empty strings and will be populated later in the script with the image IDs generated by the Docker build process. These variables are used throughout the script, allowing for easy configuration and reuse. The repository names and tag can be easily changed to point to different repositories or versions of the code without having to modify the script itself.

```
stages {
    stage('Fetch code from github') {
        steps {
            git branch: 'main',
            url: 'https://github.com/anarghya15/PeerPulse'
        }
    }
```

Figure 58: Fetching code from GitHub

The `stage` keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The “Fetch code from github” string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The git step uses the Git plugin to clone the repository located at the specified URL https://github.com/anarghya15/PeerPulse.git and checkout the master branch.

The `credentialsId` parameter specifies the ID of a Jenkins credential that contains the username and password or SSH key to authenticate with GitHub. This stage fetches the code from the GitHub repository and makes it available for building and testing in subsequent stages of the pipeline.

```
stages {
    stage('Fetch code from github') {
        steps {
            git branch: 'main',
            url: 'https://github.com/anarghya15/PeerPulse'
        }
    }
    stage('Build backend code using Maven') {
        steps {
            script{
                sh 'mvn -f peerpulse-backend clean install'
            }
        }
    }
```

Figure 59: Building the executable using Maven

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Maven Building string" is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The script block allows you to run arbitrary code in the pipeline, in this case, a shell command. The sh step runs a shell command to build the Maven project located in the **peerpulse-backend** directory using the clean install goals. This stage compiles the source code of the Maven project, runs tests, and generates the binary files (such as JAR or WAR files) that can be deployed to a production environment.

```
stage('Create backend docker image') {
    steps {
        script{
            backendImage = docker.build(backendRepositoryName + ":" + tag, "./peerpulse-backend")
        }
    }
}
stage('Push backend image to docker hub') {
    steps {
        script{
            // By default, the registry will be dockerhub
            docker.withRegistry('', 'DockerHubCred'){
                backendImage.push()
            }
        }
    }
}
```

Figure 60: Creating Docker image for the backend

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Docker image creation for backend" string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The script block allows you to run arbitrary code in the pipeline, in this case, using the Docker Pipeline plugin to build a Docker image. The `docker.build` step builds a Docker image using the Dockerfile located in the ./peerpulse-backend directory and tags it with the specified repository name and tag. This stage creates a Docker image for the backend application, which can be pushed to a container registry and deployed to a production environment. The resulting Docker image ID is stored in the `backendImage` environment variable, which can be used in subsequent stages of the pipeline.

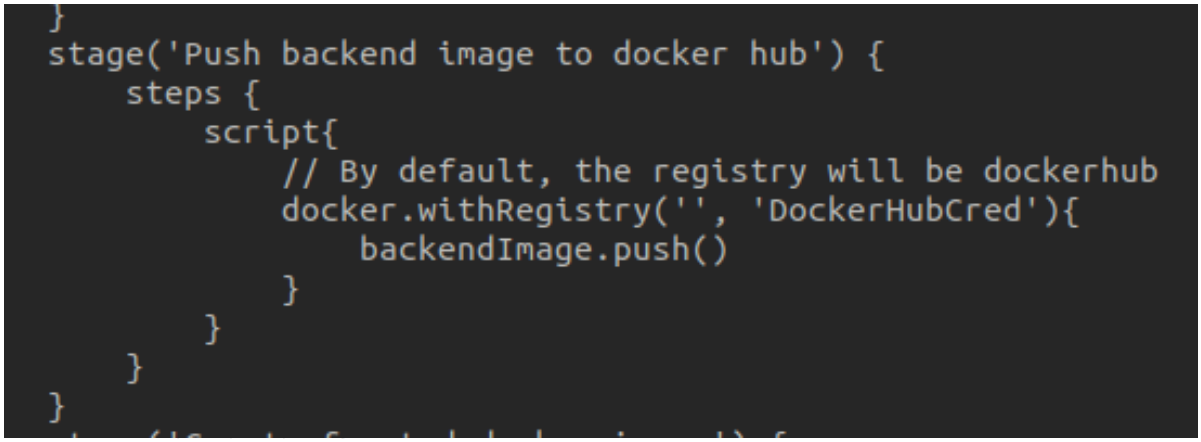



Figure 61: Pushing the created Docker image for the backend to Docker Hub

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Dockerhub backend image push" string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The script block allows you to run arbitrary code in the pipeline, in this case, using the Docker Pipeline plugin to push a Docker image to a container registry. The `docker.withRegistry` step configures the Docker Pipeline plugin to use the specified registry (`dockerhub`) and credentials (`dockerhub-credentials`) for authentication. The `backendImage.push()` step pushes the Docker image specified by the `backendImage` environment variable to the specified registry. This stage pushes the Docker image for the backend application to a container registry, making it available for deployment to a production environment.

```
stage('Create frontnd docker image') {
    steps {
        script{
            frontendImage = docker.build(frontendRepositoryName + ":" + tag, "./peerpulse-frontend")
        }
    }
}
stage('Push frontend image to docker hub') {
    steps {
        script{
            // By default, the registry will be dockerhub
            docker.withRegistry('', 'DockerHubCred'){
                frontendImage.push()
            }
        }
    }
}
```

Figure 62: Creating Docker image for the frontend

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Docker image creation for frontend" string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. remove this redundancy The script block allows you to run arbitrary code in the pipeline, in this case, using the Docker Pipeline plugin to build a Docker image. The `docker.build` step builds a Docker image using the Dockerfile located in the ./peerpulse-frontend directory and tags it with the specified repository name and tag. This stage creates a Docker image for the frontend application, which can be pushed to a container registry and deployed to a production environment. The resulting Docker image ID is stored in

the `frontendImage` environment variable, which can be used in subsequent stages of the pipeline.

```
stage('Dockerhub frontend image push')
{
    steps
    {
        script
        {
            // By default, the registry will be dockerhub
            docker.withRegistry('', 'dockerhub-credentials')
            {
                frontendImage.push()
            }
        }
    }
}
```

Figure 63: Pushing the created Docker image for the frontend to Docker Hub

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Dockerhub frontend image push" string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The script block allows you to run arbitrary code in the pipeline, in this case, using the Docker Pipeline plugin to push a Docker image to a container registry. The `docker.withRegistry` step configures the Docker Pipeline plugin to use the specified registry (`dockerhub`) and credentials (`dockerhub-credentials`) for authentication. The `frontendImage.push()` step pushes the Docker image specified by the `frontendImage` environment variable to the specified registry. This stage pushes the Docker image for the frontend application to a container registry, making it available for deployment to a production environment.

```
stage('Deploying logstash and filebeats inside the kubernetes cluster')
{
    steps
    {
        withKubeConfig([credentialsId: 'aksConfig',
        serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io'])
        {
            sh 'helm upgrade --install filebeat kubeDeploy/filebeat'
            sh 'helm upgrade --install logstash kubeDeploy/logstash'
        }
    }
}
```

Figure 64: Deploying Logstash and Filebeat inside the Kubernetes cluster

The stage keyword defines a stage in the pipeline, which represents a logical division of work in the pipeline. The "Deploying logstash and filebeats inside the kubernetes cluster" string is the name of the stage, which is displayed in the Jenkins UI. The steps keyword defines the steps that are executed in the stage. In this case, there is only one step. The `withKubeConfig` step allows you to specify the Kubernetes cluster to use for the deployment, along with authentication credentials. The sh step runs a shell command to deploy Logstash and Filebeat using Helm. This

stage deploys Logstash and Filebeat to a Kubernetes cluster, allowing them to collect and process logs from the backend and frontend applications. The Logstash and Filebeat configurations are specified in the `kubeDeploy/logstash` and `kubeDeploy/filebeat` directories, respectively. The `helm upgrade -install` command installs the specified Helm chart or upgrades it if it already exists.

```
stage('Adding secrets and config maps to kubernetes cluster')
{
    steps
    {
        withKubeConfig([credentialsId: 'aksConfig',
        serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io'])
        {
            sh 'kubectl apply -f kubeDeploy/mysql-root-credentials.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-credentials.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-configmap.yml'
        }
    }
}
```

Figure 65: Adding secrets and config maps to the Kubernetes cluster

The `withKubeConfig` step allows you to specify the Kubernetes cluster to use for the deployment, along with authentication credentials. The sh step runs a shell command to apply the specified YAML files to the Kubernetes cluster. This stage adds three secrets and config maps to the Kubernetes cluster, which are used by the MySQL database:

- `mysql-root-credentials.yml`: Contains the root password for the MySQL database.
- `mysql-credentials.yml`: Contains the username and password for the MySQL database user.
- `mysql-configmap.yml`: Contains configuration settings for the MySQL database, such as the database name and character set.

The `kubectl apply -f` command applies the YAML files to the Kubernetes cluster, creating or updating the specified resources.

```
stage('Deleting older application deployment')
 {
    steps
    {
        withKubeConfig([credentialsId: 'aksConfig',
        serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io'])
        {
            sh 'kubectl delete -f kubeDeploy/mysql-deployment.yml'
            sh 'kubectl delete -f kubeDeploy/backend-deployment.yml'
            sh 'kubectl delete -f kubeDeploy/frontend-deployment.yml'
        }
    }
}
```

Figure 66: Deleting the older deployments of the applications

The `withKubeConfig` step allows you to specify the Kubernetes cluster to use for the deployment, along with authentication credentials. The sh step runs a shell command to delete the specified Kubernetes deployments. This stage deletes the older deployments of the MySQL database, backend application, and frontend application from the Kubernetes cluster. This is necessary to ensure that there is no conflict with the new deployments and that the resources are cleaned up properly. The `kubectl delete -f` command deletes the Kubernetes resources specified in the YAML files.

```
stage('Deploying application to kubernetes cluster')
{
    steps
    {
        withKubeConfig([credentialsId: 'aksConfig',
        serverUrl: 'https://gradeus-dns-9q6us9tx.hcp.eastus.azmk8s.io'])
        {
            sh 'kubectl apply -f kubeDeploy/mysql-service.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-pvc.yml'
            sh 'kubectl apply -f kubeDeploy/mysql-deployment.yml'
            sh 'kubectl apply -f kubeDeploy/backend-service.yml'
            sh 'kubectl apply -f kubeDeploy/backend-deployment.yml'
            sh 'kubectl apply -f kubeDeploy/frontend-service.yml'
            sh 'kubectl apply -f kubeDeploy/frontend-deployment.yml'
        }
    }
}
```

Figure 67: Deploying the application to the Kubernetes cluster

The `withKubeConfig` step allows you to specify the Kubernetes cluster to use for the deployment, along with authentication credentials. The sh step runs a shell command to apply the specified YAML files to the Kubernetes cluster. This stage deploys the MySQL database, backend application, and frontend application to the Kubernetes cluster. The resources are deployed in the following order:

- `mysql-service.yml`: Creates a Kubernetes service for the MySQL database.
- `mysql-pvc.yml`: Creates a Kubernetes persistent volume claim for the MySQL database.
- `mysql-deployment.yml`: Deploys the MySQL database to the Kubernetes cluster.
- `backend-service.yml`: Creates a Kubernetes service for the backend application.
- `backend-deployment.yml`: Deploys the backend application to the Kubernetes cluster.
- `frontend-service.yml`: Creates a Kubernetes service for the frontend application.
- `frontend-deployment.yml`: Deploys the frontend application to the Kubernetes cluster.

The `kubectl apply -f` command applies the YAML files to the Kubernetes cluster, creating or updating the specified resources.

## 14.1 Ansible

**Ansible is also used as an alternative method of deployment.**

Ansible is an open-source automation tool that simplifies configuration management, application deployment, and task automation. It is agentless, meaning it doesn't require any software to be installed on the target system, making it easy to use and deploy across a wide range of environments.

At its core, Ansible works by using SSH to connect to remote systems and execute tasks. These tasks are defined in playbooks, which are YAML files that specify a set of actions to be performed on one or more systems. Playbooks can be used to perform a wide range of tasks, including installing packages, configuring services, and deploying applications.

Ansible is designed to be easy to learn and use, with a simple syntax and a large collection of modules and plugins that make it easy to extend its functionality. It also features a powerful templating system, which allows users to dynamically generate configuration files and other resources based on variables and conditions.

One of the key benefits of Ansible is its ability to manage systems at scale. It can be used to manage thousands of systems at once, and its "push" model means that changes can be made quickly and easily, without the need for complex orchestration or coordination.

Another key feature of Ansible is its support for idempotent operations. This means that if a task has already been performed, Ansible will not perform it again, unless explicitly instructed to do so. This helps ensure that systems remain in a consistent state, even when changes are made over time.

Ansible is a powerful and flexible automation tool that can help simplify the management of complex IT environments. It is widely used in DevOps, system administration, and cloud computing, and is supported by a large and active community of users and contributors.

In Ansible, a playbook is a YAML file that defines a set of tasks to be performed on one or more systems. Playbooks are the heart of Ansible, and they provide a way to automate complex tasks and workflows in a declarative and repeatable manner.

A playbook consists of one or more plays, where each play is a sequence of tasks that are executed on a specific set of hosts. Each play also contains a set of variables, which are used to define the state of the system and to control the behavior of the tasks.

The main parts of a playbook are as follows:

- **Hosts**: This section specifies the hosts or groups of hosts that the playbook will be executed against. Hosts can be specified using a variety of methods, including IP address, hostname, or group name.
- **Variables**: This section defines the variables that will be used by the tasks in the playbook. Variables can be defined at the playbook level, at the play level, or at the task level, and they can be set to static values or dynamically generated using Jinja2 templates.
- **Tasks**: This section defines the tasks that will be executed on the hosts specified in the "hosts" section. Tasks can perform a wide range of actions, including installing packages, configuring services, and copying files. Tasks can also be organized into roles, which are reusable collections of tasks that can be shared across multiple playbooks.
- **Handlers**: This section defines the handlers that will be executed when triggered by a task. Handlers are similar to tasks, but they are only executed when explicitly triggered, and they are used to perform actions such as restarting services or reloading configuration files.
- **Variables** (again): This section defines additional variables that will be used by the handlers.
- **Tags**: This section defines tags that can be used to selectively execute specific tasks or groups of tasks within the playbook.

- **Conditionals**: This section defines conditionals that can be used to selectively execute specific tasks or groups of tasks within the playbook based on the state of the system.

Playbooks provide a powerful and flexible way to automate complex tasks and workflows in a consistent and repeatable manner, and they are a core feature of the Ansible automation tool.

```
// stage('Ansible Deployment') {
//     steps {
//       ansiblePlaybook installation: 'Ansible', playbook: 'deploy-playbook.yml'
//     }
// }
```

Figure 68: Ansible step in the Jenkins pipeline script

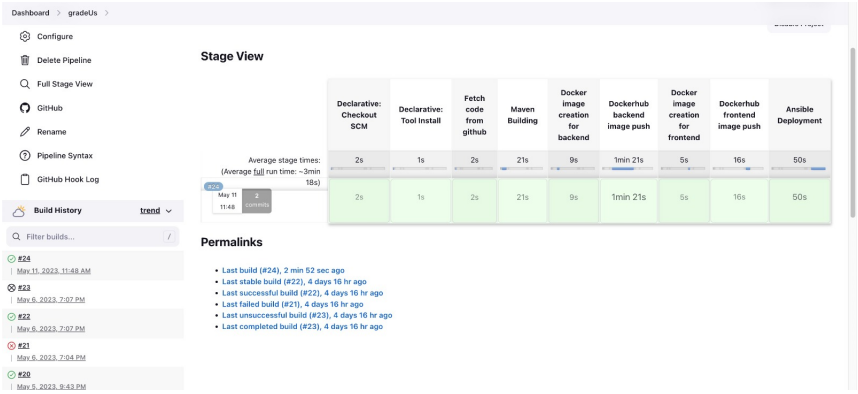



Figure 69: Jenkins build with the Ansible step

application.properties deploy-playbook.yml

```yaml
---
- name: "Deploying on the kubernetes cluster"
  hosts: localhost
  connection: local
  environment:
    KUBECONFIG: /var/lib/jenkins/.kube/config

  tasks:
  - name: Deleting older application deployment
    ansible.builtin.shell: |
      kubectl delete -f kubeDeploy/mysql-deployment.yml --ignore-not-found=true
      kubectl delete -f kubeDeploy/backend-deployment.yml --ignore-not-found=true
      kubectl delete -f kubeDeploy/frontend-deployment.yml --ignore-not-found=true

  - name: Deploying logstash and filebeats inside the kubernetes cluster
    ansible.builtin.shell: |
      helm upgrade --install filebeat kubeDeploy/filebeat
      helm upgrade --install logstash kubeDeploy/logstash

  - name: Adding secrets to kubernetes cluster
    ansible.builtin.shell: |
      kubectl apply -f kubeDeploy/mysql-root-credentials.yml
      kubectl apply -f kubeDeploy/mysql-credentials.yml

  - name: Adding config map to kubernetes cluster
    ansible.builtin.shell: kubectl apply -f kubeDeploy/mysql-configmap.yml

  - name: Deploying application to kubernetes cluster
    ansible.builtin.shell: |
      kubectl apply -f kubeDeploy/mysql-pvc.yml
      kubectl apply -f kubeDeploy/mysql-deployment.yml
      kubectl apply -f kubeDeploy/mysql-service.yml
      kubectl apply -f kubeDeploy/backend-deployment.yml
```

Figure 70: A snippet of the Ansible playbook

This Ansible playbook deploys an application to a Kubernetes cluster.

- The first line "—" indicates the start of a YAML file.
- The "name" key under the first list item is a description of the play.

- The “hosts” key specifies that this playbook will be executed on the localhost.
- The “connection” key specifies that the connection to the host will be local.
- The “tasks” key is the list of tasks that will be executed in the playbook.
- The first task is commented out and is not executed. It is an example of how to use the Ansible shell module to run the Azure CLI login command.
- The second task uses the shell module to upgrade or install Helm charts for Filebeat and Logstash in the Kubernetes cluster.
- The third task adds secrets and config maps to the Kubernetes cluster using the `kubectl apply` command.
- The fourth task deletes older application deployments using the `kubectl delete` command.
- The fifth task deploys the application to the Kubernetes cluster using the `kubectl apply` command.
- The “environment” key is used to specify the location of the KUBECONFIG file, which is used to authenticate with the Kubernetes cluster.

The playbook automates the deployment of an application to a Kubernetes cluster using a set of tasks that are executed in a specific order. It demonstrates how Ansible can be used to automate complex workflows and tasks in a declarative and repeatable manner.

# 15 Application Screenshots

## 15.1 Instructor-side

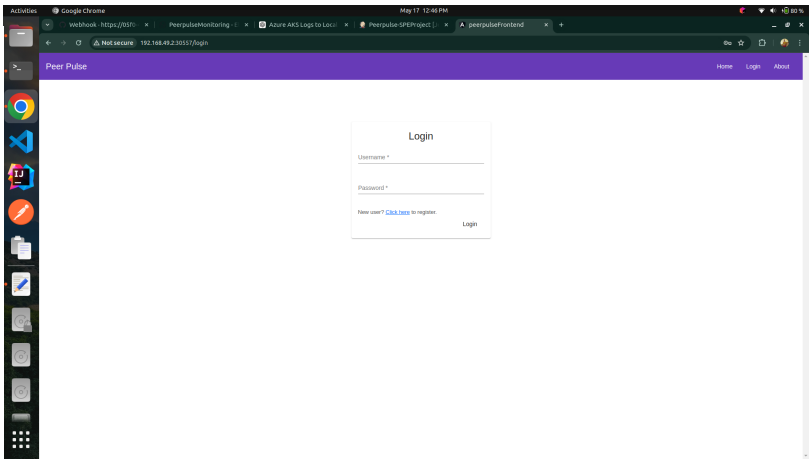



Figure 71: The login page of the application

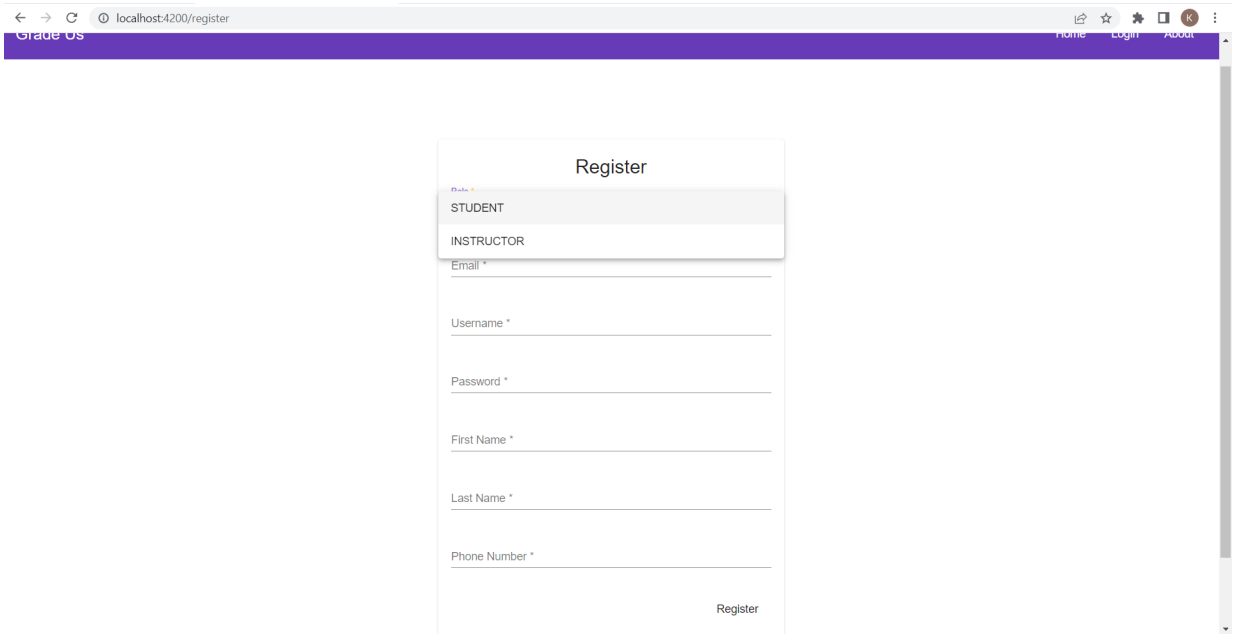



Figure 72: Th register page of the application, as student or instructor

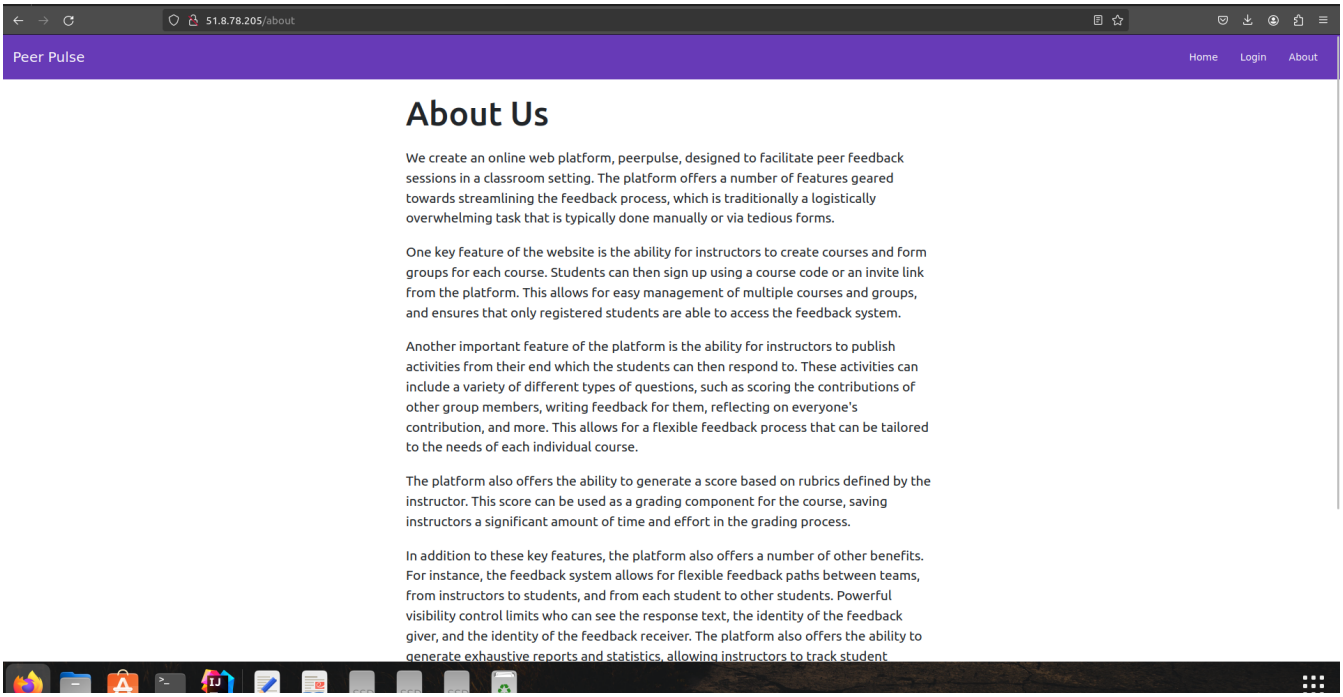



Figure 73: The about page of the application

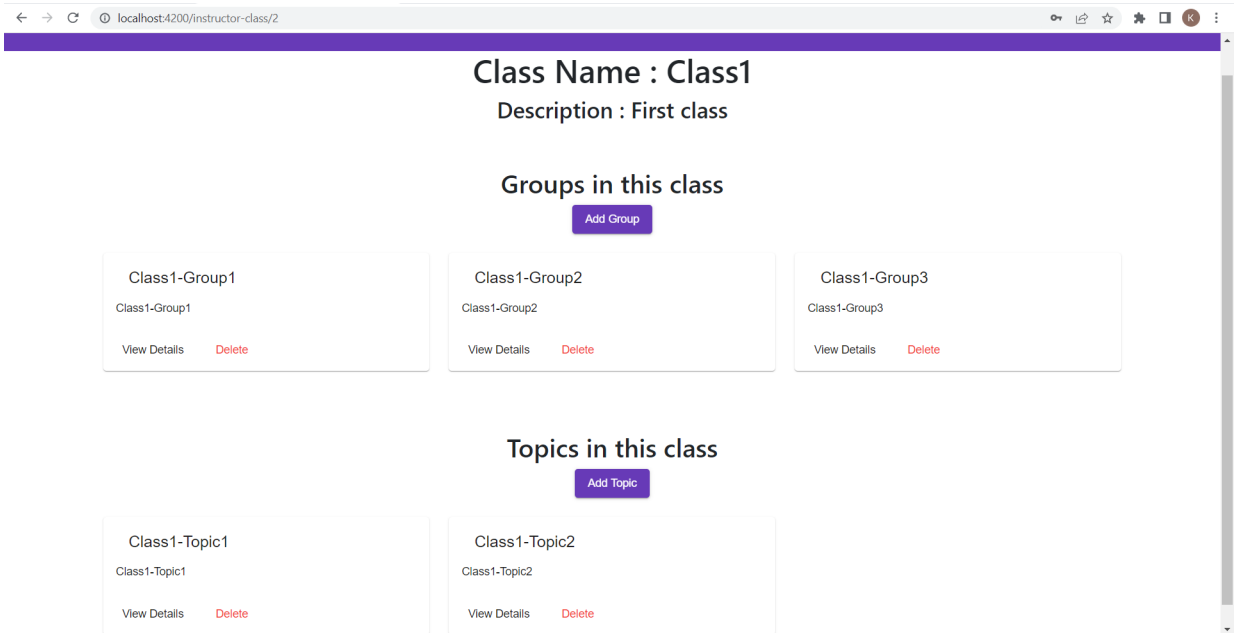



Figure 74: Dashboard for the instructor

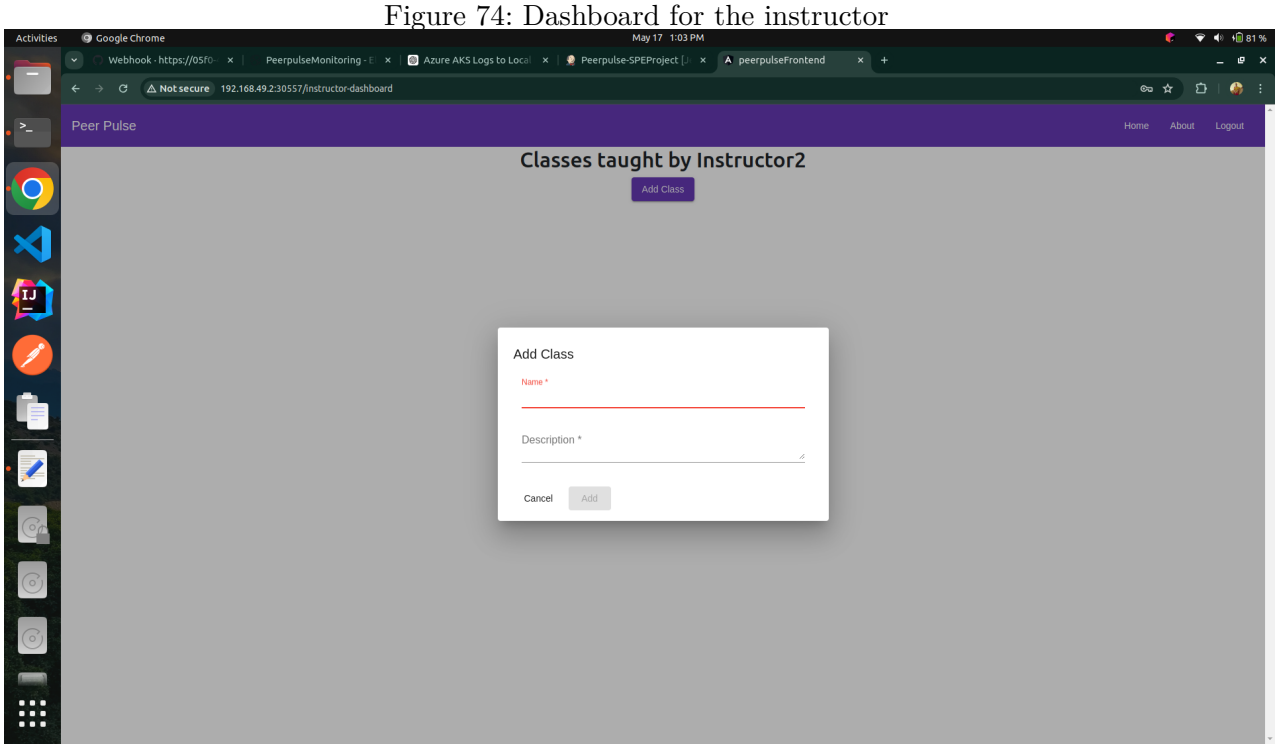



Figure 75: Adding a class on the instructor dashboard

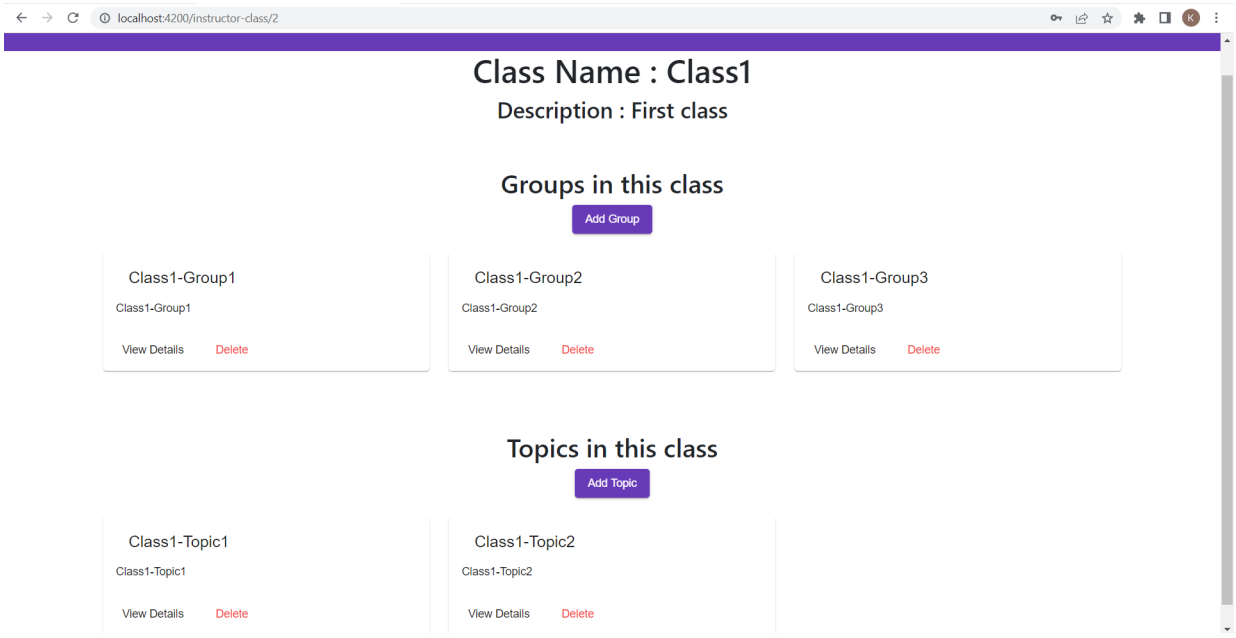



Figure 76: Dashboard of a class for the instructor

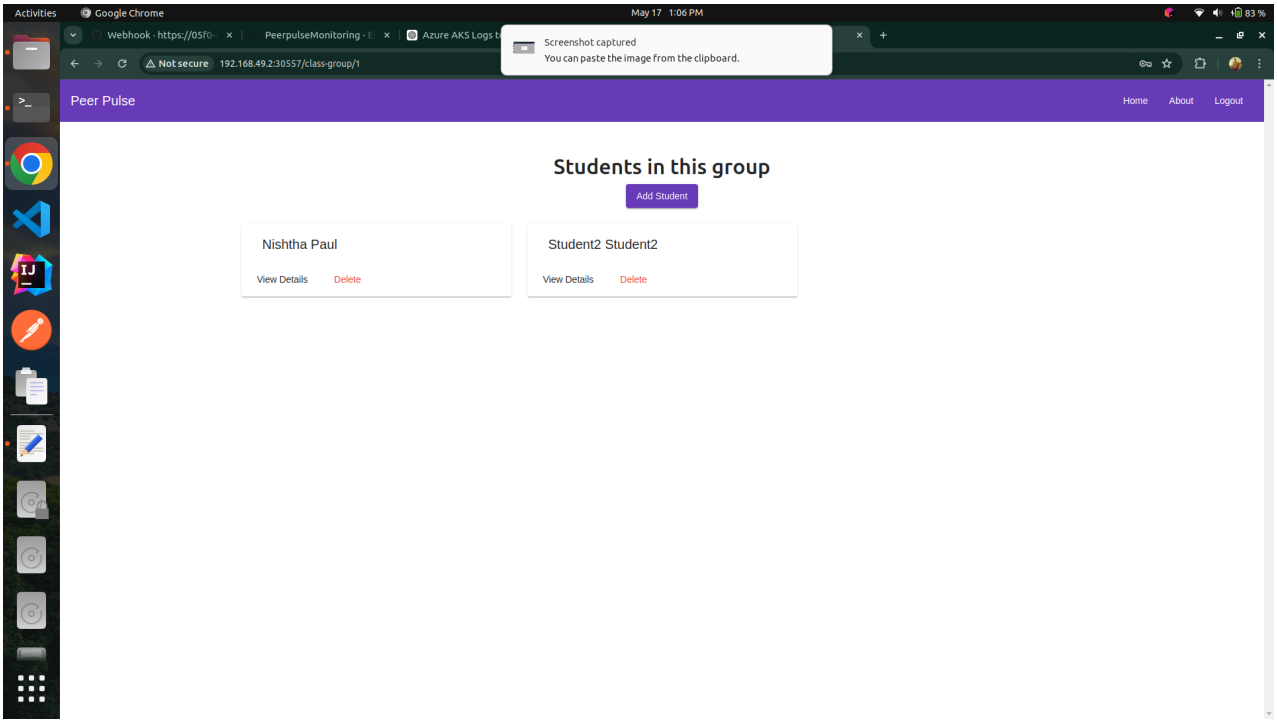



Figure 77: Adding students to a group by the instructor

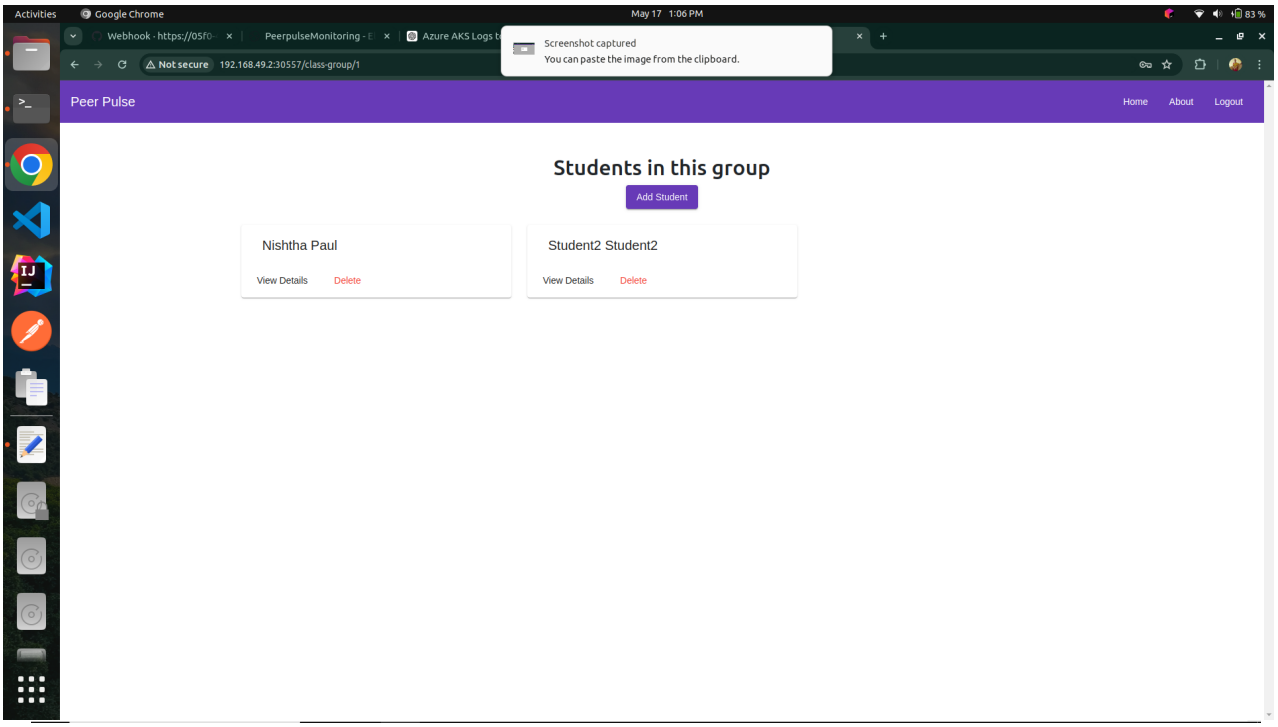



Figure 78: Group dashboard for the instructor

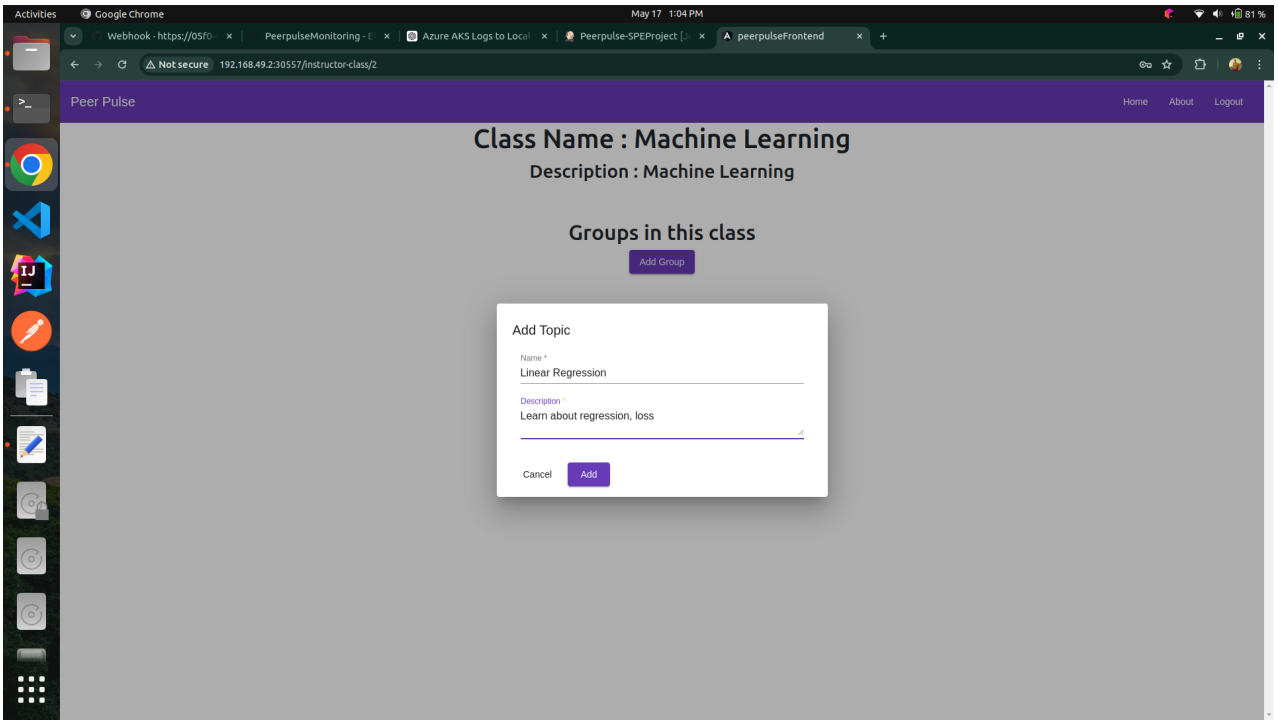



Figure 79: Adding a topic to a class by the instructor

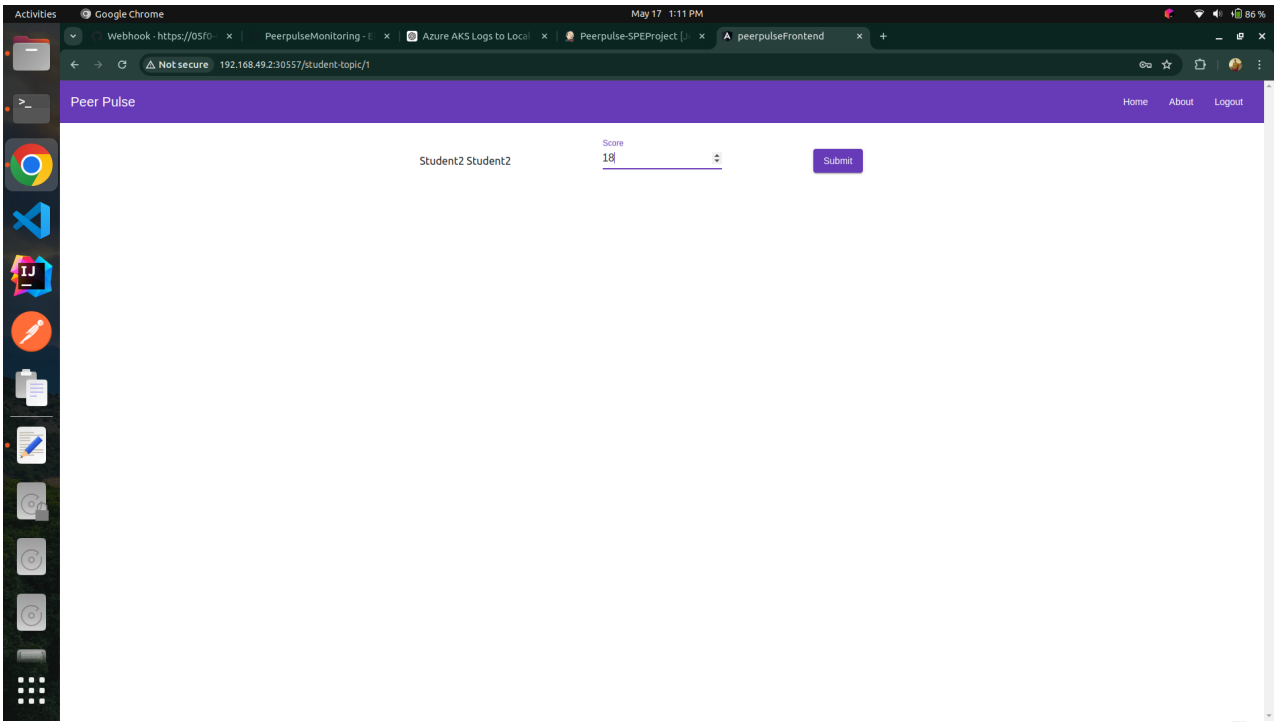



Figure 80: The visibility of the marks given by the students to each other in a topic to the instructor

## 15.2 Student-side

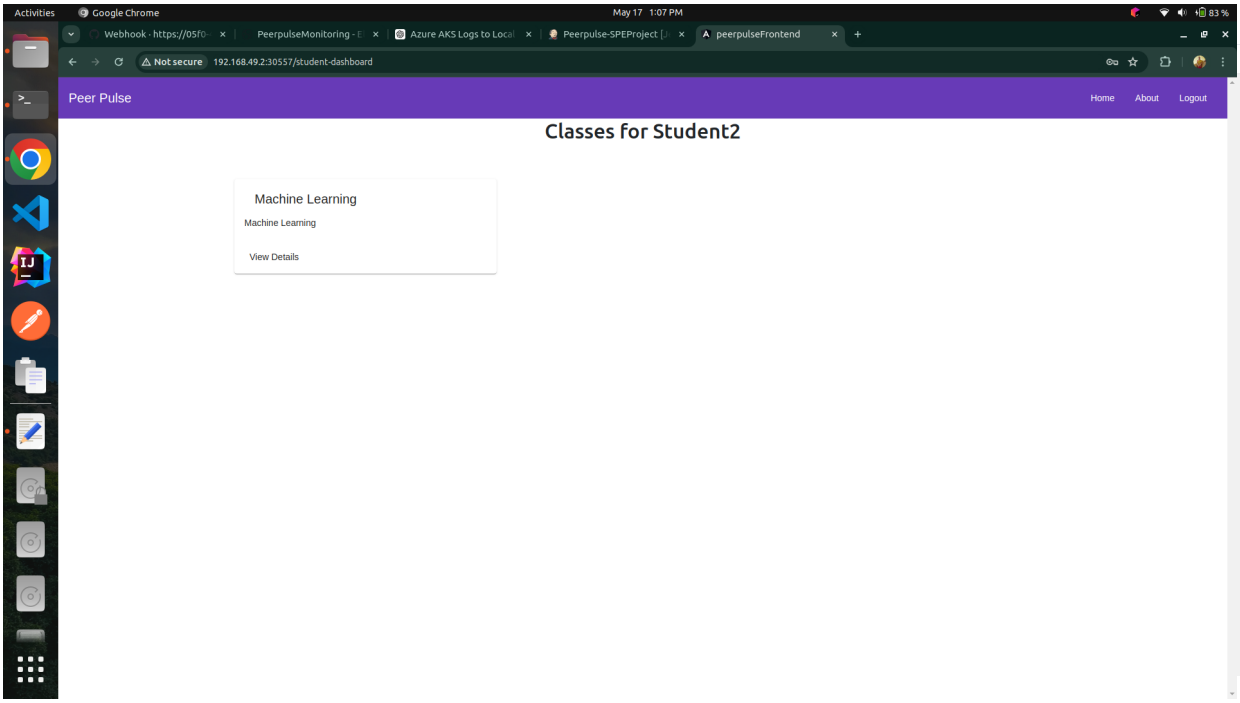



Figure 81: The dashboard for a student

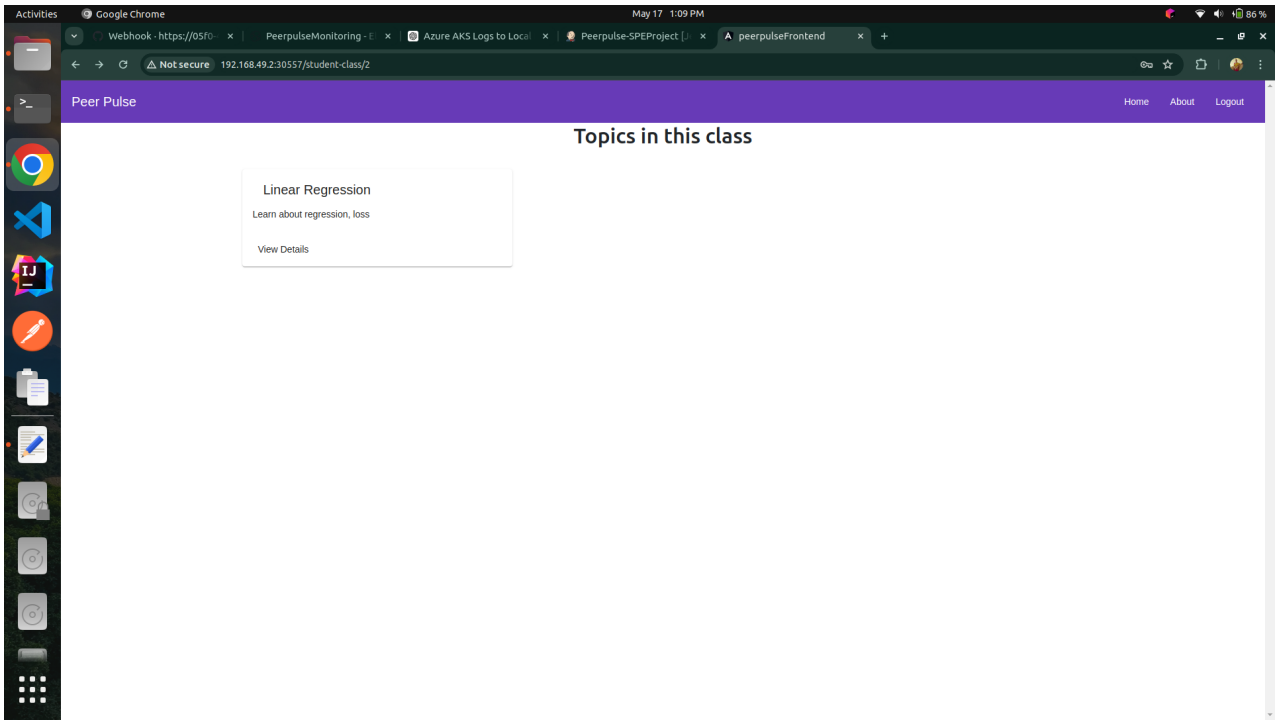



Figure 82: The class dashboard for a student

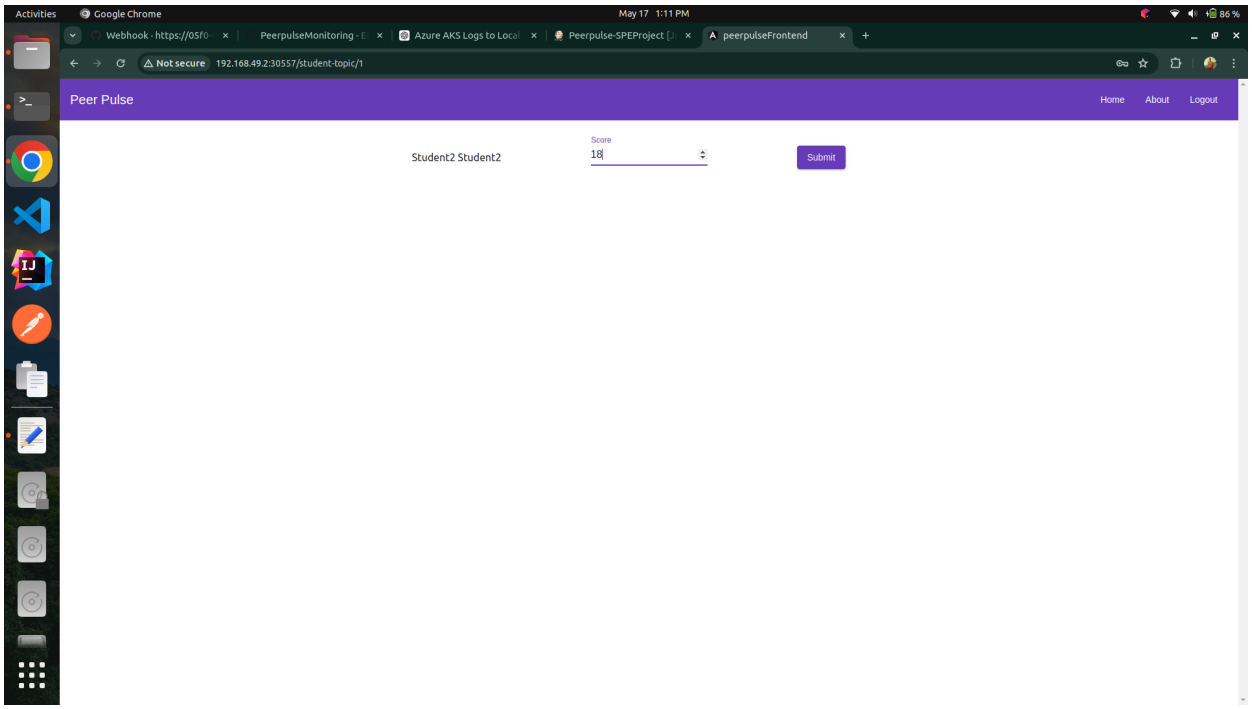



Figure 83: Assignment of marks to others by a student

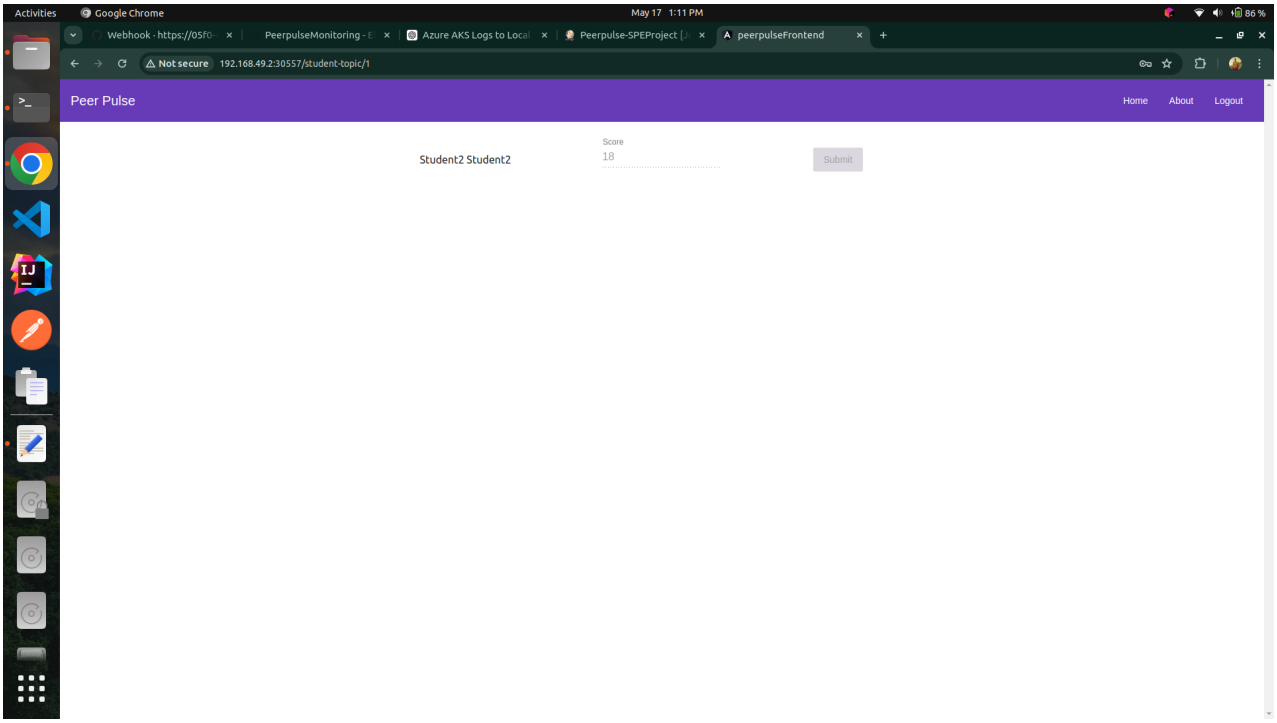



Figure 84: Marks, once submitted, as frozen and cannot be changed by the student, without deletion from the instructor

## 16 API Documentation

API documentation, also known as Application Programming Interface documentation, is a set of guidelines, instructions, and examples that provide developers with the necessary information to interact with a particular software application or web service using its API.

An API is essentially a set of protocols, tools, and routines that enable different software applications to communicate with each other. APIs allow developers to create applications that can interact with other applications, services, and data sources, which can greatly enhance the functionality and usefulness of their software.

API documentation is an essential resource for developers who want to integrate with an API and build applications that can access and utilize the functionality of a software application or web service. It provides clear and concise instructions on how to use the API, along with examples and best practices that help developers avoid common pitfalls and optimize their code for performance and reliability.

Swagger is an open-source software framework that is used to design, build, document, and test RESTful web services. It provides developers with a set of tools and libraries that enable them to create APIs that are easy to understand and use, both by humans and machines.


At its core, Swagger uses a specification language called OpenAPI (formerly known as Swagger Specification) to define the structure of RESTful APIs. OpenAPI is a machine-readable format that describes the various endpoints, parameters, and responses of an API in a way that can be easily consumed by other software applications.

Swagger provides a range of tools and features to help developers work with OpenAPI specifications, including:

- **Swagger Editor**: A web-based editor that allows developers to create, edit, and validate OpenAPI specifications.
- **Swagger UI**: A web-based user interface that displays the documentation for an API in an interactive and user-friendly way.
- **Swagger Codegen**: A tool that generates server-side and client-side code based on an OpenAPI specification, allowing developers to quickly build API implementations in a variety of programming languages.
- **Swagger Inspector**: A tool that allows developers to test and debug their APIs by sending sample requests and responses.

By using Swagger, developers can create APIs that are not only well-documented and easy to use, but also conform to industry best practices and standards. This can help to improve the overall quality and reliability of the APIs, as well as reduce the time and effort required to build and maintain them.

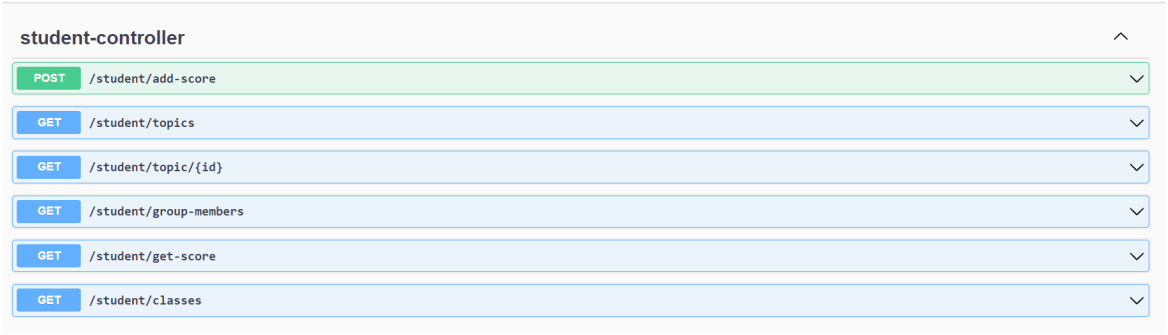



Figure 85: APIs for the `student-controller`


Figure 86: APIs for the `instructor-controller`

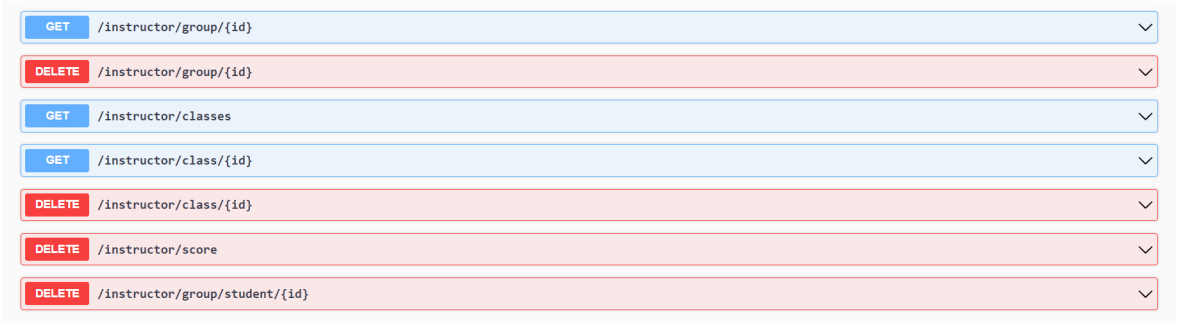



Figure 87: APIs for the `inctructor-controller`, contd

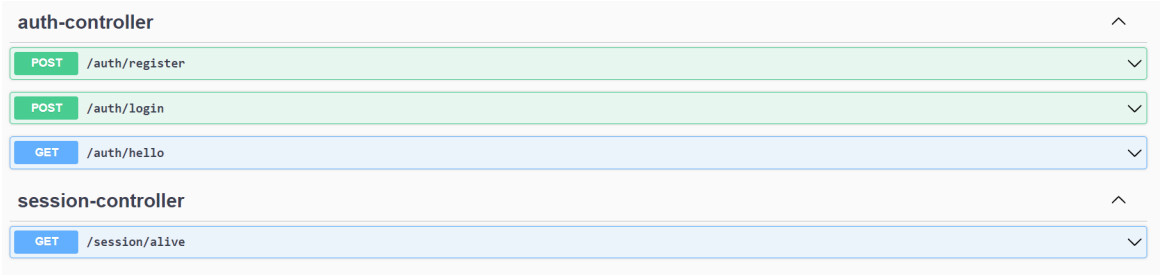



Figure 88: APIs for the `auth-controller` and `session-controller`

# 17 Future Work

- Students signing up using a course code or an invite link from the platform.
- Different type of activities such as different types of questions, scoring the contributions of other group members, writing feedback for them, reflecting on everyone's contribution, and more. This allows for a flexible feedback process that can be tailored to the needs of each individual course.
- Allowing for flexible feedback paths between teams, from instructors to students, and from each student to other students.
- Powerful visibility control limits who can see the response text, the identity of the feedback giver, and the identity of the feedback receiver.
- The ability to generate exhaustive reports and statistics, allowing instructors to track student progress over time.
- A wide range of question types, including essay questions, MCQ questions, multiple select questions, numeric scale questions, and questions measuring contribution in team projects. This ensures that the feedback system is flexible enough to accommodate a wide range of different types of courses and assignments, making it a valuable tool for instructors across a range of different disciplines.

# 18 Challenges

- **Configuring the reverse proxy and accessing the backend using dns.**

  Initially, while testing on minikube, we exposed the backend service using nodePort. Now to get the url for api calls in frontend, we hardcoded the minikube url and port mentioned in backend service to make api calls.

Afterwards, we decided to only expose the frontend service and call backend apis using domain name internally exposed by the backend service.

Directly writing the domain name in the frontend code didn't work as browser is not inside the cluster and it can't identify the domain name.

To solve this, we reverse proxy using nginx. Which maps the urls to the backend using the domain name. And as this nginx pod is running inside the cluster, it can identify and translate this domain name to get to the backend service.

```
upstream Backend {
    # hello is the internal DNS name used by the backend Service inside Kubernetes
    server backend-deployment:8500;
}

server {
    listen 80;
    location / {
        root /usr/share/nginx/html;
        try_files $uri /index.html;
    }

    location /app/ {
        proxy_pass http://Backend/;
    }
}
```

Now, while making the backend api calls, we need to just send requests to '/app'.

```
5 lines (5 sloc)  | 112 Bytes
export const environment = {
  production: true,
  // rootUrl: "http://192.168.64.7:30163"
  rootUrl: "/app"
};
```

- **Mysql version and query difference.**

  In the local development environment, the version of myql that we were using was 8.33 whereas in the mysql image, the version we used was 5.7. We wrote an SQL query with INTERSECTION in it, it was working in the local environment but not when deployed on the cluster because MySQL 5.7 doesn't support INTERSECTION. Therefore we had to change the query according to the 5.7 version.

- **Jenkins docker problem in mac.**

  In mac, docker command was not directly working with Jenkins. Needed to change some xml configuration file in Jenkins.

- **Minikube not working with lesser memory.**

  Initially we tried to run minikube with 2GB ram. We were not able to build the angular application. It was failing and stopping abruptly. We fixed it by allotting 4GB ram to it.

# 19 Conclusion

This DevOps project that was built using Spring Boot, Angular, MySQL, Git, GitHub, Jenkins, Docker, Kubernetes, Azure, and AWS is a prime example of how DevOps can transform the software development and deployment process. DevOps is a methodology that combines software development and IT operations in order to deliver high-quality software faster and more efficiently. This project demonstrates how a range of powerful tools and platforms can be used to build and deploy modern, cloud-native applications in a fast and efficient way.

The project was built using Spring Boot, a popular Java-based framework for building web applications, and Angular, a popular JavaScript framework for building user interfaces. These technologies were used to build a modern, cloud-native application that is highly scalable and resilient. The application was backed by a MySQL database, which provided a reliable and scalable data storage solution.

Git and GitHub were used for version control and collaboration. Git is a distributed version control system that allows developers to track changes to their code over time, while GitHub is a cloud-based platform that provides a range of collaboration and workflow tools for software development teams. By using Git and GitHub, the project team was able to work together to build high-quality software in a collaborative way.

Jenkins was used for continuous integration and continuous deployment (CI/CD). Jenkins is an open-source automation server that is used to automate the building, testing, and deployment of software applications. By using Jenkins, the project team was able to automate the entire software development process, from code changes to deployment.

Docker was used for containerization. Docker is a platform that allows developers to build, ship, and run applications in containers. Containers are lightweight, portable, and self-contained environments that can be easily deployed across different platforms and environments. By using Docker, the project team was able to build and deploy the application in a consistent and reliable way.

Kubernetes was used for container orchestration. Kubernetes is an open-source platform for managing containerized workloads and services. By using Kubernetes, the project team was able to manage the containers that were running the application in a scalable and efficient way.

Azure and AWS were used for cloud hosting. Azure and AWS are two of the most popular cloud platforms for hosting applications. By using these platforms, the project team was able to deploy the application in a highly available and scalable way, while also taking advantage of the many other services and tools that these platforms provide.

Overall, the DevOps project built using Spring Boot, Angular, MySQL, Git, GitHub, Jenkins, Docker, Kubernetes, Azure, and AWS demonstrates the power and flexibility of the DevOps approach to software development and deployment. By leveraging a range of powerful tools and platforms, the project shows how DevOps can streamline the software development process, making it faster, more efficient, and more reliable.

The project also demonstrates how a modern, cloud-native architecture can be used to build scalable and resilient applications, while automation tools like Jenkins and Docker Compose can help to simplify the process of building, testing, and deploying those applications. By using Git and GitHub for version control and collaboration, the project shows how teams can work together to build high-quality software, while Kubernetes and cloud platforms like Azure and AWS provide powerful tools for managing and scaling complex applications in production.

In conclusion, the DevOps project built using Spring Boot, Angular, MySQL, Git, GitHub, Jenkins, Docker, Kubernetes, Azure, and AWS provides a compelling example of how DevOps can be used to build and deploy modern, cloud-native applications in a fast and efficient way. The combination of powerful tools and platforms, along with a focus on collaboration, automation, and continuous improvement, makes DevOps an essential approach for any organization that wants to stay competitive in today's fast-paced software development landscape.

## 20 References


1. "The Top 10 DevOps Trends to Watch in 2022" by TechBeacon: https://techbeacon.com/devops
2. "DevOps and Agile: What's the Difference?" by Atlassian: https://www.atlassian.com/agile/devops
3. "The Rise of NoOps: The End of Operations?" by The New Stack: https://thenewstack.io/the-rise-of-noops-the-end-of-operations/
4. "The 2021 State of DevOps Report" by Puppet: https://puppet.com/resources/report/state-of-devops-report/
5. "What is GitOps, and Why is it Gaining Traction in Cloud Native?" by CloudBees: https://www.cloudbees.com/gitops/what-is-gitops
6. "The DevOps Handbook - Summary and Key Takeaways" by DevOpsGroup: https://www.devopsgroup.com/2019/05/24/the-devops-handbook-summary-and-key-takeaways/
7. "What is Infrastructure as Code?" by HashiCorp: https://www.hashicorp.com/resources/what-is-infrastructure-as-code
8. "A Beginner's Guide to Kubernetes" by CircleCI: https://circleci.com/blog/beginners-guide-to-kubernetes/
9. "DevOps Best Practices: Continuous Integration and Continuous Deployment" by XebiaLabs: https://blog.xebialabs.com/2018/02/08/devops-best-practices-ci-cd/
10. "What is Site Reliability Engineering (SRE)?" by Red Hat Developer Blog: https://developers.redhat.com/blog/2019/12/06/what-is-site-reliability-engineering-sre/
11. "The Role of DevOps in Digital Transformation" by Inedo Blog: https://inedo.com/blog/the-role-of-devops-in-digital-transformation
12. "What is Jenkins? The CI/CD Server Explained" by Jenkins.io: https://www.jenkins.io/doc/book/introduction/what-is-jenkins/
13. "A Beginner's Guide to Infrastructure Automation with Terraform" by AWS DevOps Blog: https://aws.amazon.com/blogs/devops/a-beginners-guide-to-infrastructure-automation-wit
14. "7 Key DevOps Metrics to Track" by Puppet Blog: https://puppet.com/blog/7-key-devops-metrics-
15. "What is DevSecOps? An Introduction to Security in DevOps" by TechBeacon: https://techbeacon.com/security/what-devsecops-introduction-security-devops
16. "The Importance of Automation in DevOps" by The New Stack: https://thenewstack.io/the-importance-of-automation-in-devops/
17. "GitOps vs. DevOps: What's the Difference?" by Weaveworks Blog: https://www.weave.works/blog/gitops-vs-devops-whats-the-difference
18. "What is ChatOps? A Comprehensive Guide to ChatOps" by VictorOps: https://victorops.com/blog/what-is-chatops-a-comprehensive-guide-to-chatops
19. "DevOps and the Cloud: A Match Made in Heaven" by Cloud Academy Blog: https://cloudacademy.com/blog/devops-and-the-cloud-a-match-made-in-heaven/

20. “The DevOps Engineer’s Toolbox: Essential Tools for DevOps Success” by The New Stack:

    https://thenewstack.io/the-devops-engineers-toolbox-essential-tools-for-devops-success

21. Techno Town Techie. (2020, September 5). Full-Stack with Angular 8 + Spring Boot + Mysql CRUD API application on Kubernetes cluster [Video]. YouTube. https://www.youtube.com/watch?v=aPzpsfQtlKY