**Web Application Deployment and Database Migration**

**Group Name: Group F**

|  |  |
| --- | --- |
| Semester | **F2023** |
| Course Code | CBD 3375 |
| Section | Section 1 |
| Project Title | **Web Application Deployment and Database Migration** |
| Group Name | **Group F** |
| Student names/Student IDs | Arsha Mary Issac (C0870815), Jyothi Maria Saji (C0872794), Nimmy George (C0872805) |
| Faculty Supervisor | William Pourmajidi |

**Submission date: *04-12-2023***

Contents

[Abstract 3](#_Toc152605240)

[Introduction 3](#_Toc152605241)

[Methods 4](#_Toc152605242)

[**Source** 5](#_Toc152605243)

[**Web Development and Database Integration:** 6](#_Toc152605244)

[**Build** 8](#_Toc152605245)

[**Application Dockerization** 8](#_Toc152605246)

[**Docker Hub Image Push** 11](#_Toc152605247)

[**Deploy** 11](#_Toc152605248)

[**Azure Virtual Machine Creation** 11](#_Toc152605249)

[**CI/CD Pipeline** 14](#_Toc152605250)

[**Database Migration** 18](#_Toc152605251)

[**AWS Server Integration** 18](#_Toc152605252)

[Results 22](#_Toc152605253)

[Conclusions and Future Work 24](#_Toc152605254)

[References 24](#_Toc152605255)

# **Abstract**

The goal of the project is to construct a website that displays current company data using JavaScript and NodeJS. We will employ a CI/CD pipeline, which includes four critical stages: source, build, deploy and migrating the database to enable effective software delivery. The source code will be uploaded to a version control system (GitHub) and tagged with the relevant version. Jenkins will be used as an automated pipeline tool to accomplish continuous integration, which comprises steps such as compiling, building, validating and then after moving the program to the staging server, we will deploy it on Azure using a docker container.

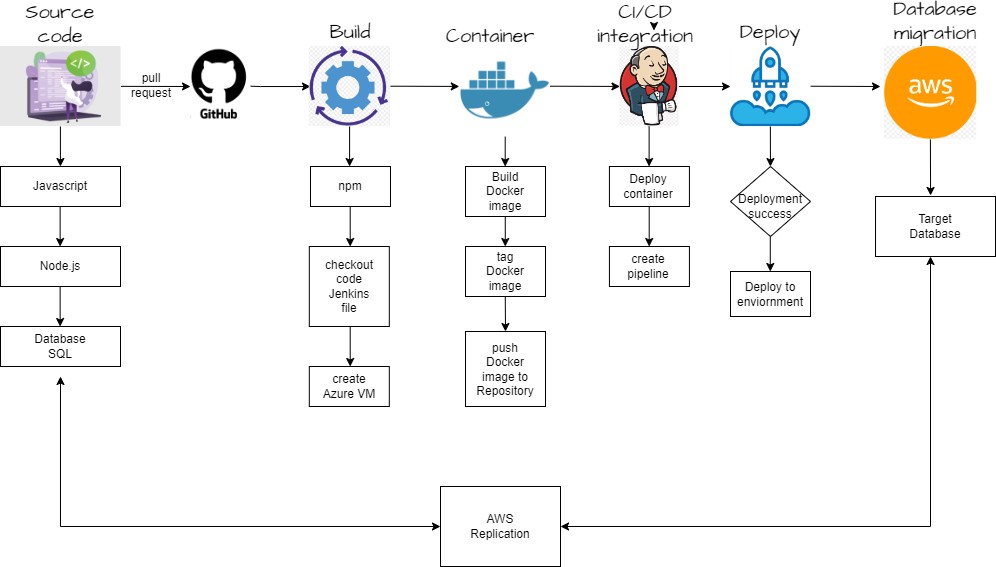
This project describes a concept to automate containerized application deployment while handling database migrations. Creating a prototype web application, posting its source code on GitHub, deploying it using Docker on an Azure Virtual Machine (VM), utilizing SQL Database as the backend, and establishing a Continuous Integration/Continuous Deployment (CI/CD) pipeline for continuous updates. It also requires migrating the SQL database to AWS using the AWS Database Migration Service. The project's goal is to demonstrate how to deploy a web application in a containerized environment, leverage cloud infrastructure, employ CI/CD approaches, and transfer a database between cloud providers. The results can benefit developers, cloud engineers, and organizations wishing to modernize their application deployment and database management practices.

# **Introduction**

Adoption of containerization, cloud infrastructure, and effective CI/CD practices has become critical in the quickly expanding environment of software development and deployment for assuring agility, scalability, and maintainability. This project aims to solve these modern difficulties by delivering a full solution for automating containerized application deployment, as well as seamless database migration management. This initiative's major goal is to demonstrate a feasible implementation of delivering an example web application in a containerized environment. This project shows the combination of cutting-edge technology to expedite the deployment process, utilizing Docker for containerization and Azure Virtual Machines for hosting. Furthermore, the use of SQL Database as the backend guarantees a solid basis for data administration.

A Continuous Integration/Continuous Deployment (CI/CD) pipeline is constructed to demonstrate how automated workflows may support quick and error-free modifications to the deployed application. This not only helps engineers preserve code integrity, but it also lays the groundwork for effective cooperation and continual development. The inclusion of the AWS Database Migration Service, which highlights the ability to effortlessly move a SQL database to AWS, is a unique component of this project. This tackles a real-world scenario in which organizations may need to shift between cloud platforms, demonstrating the suggested solution's flexibility and interoperability. This project is a helpful resource for developers, cloud engineers, and organizations looking to modernize their application deployment and database management practices. Stakeholders may get insights into best practices for containerization, cloud infrastructure utilization, CI/CD deployment, and cross-cloud database transfer by following the methods indicated.

# **Methods**



*figure 1: Methodology*

Pipelines for Continuous Integration and Continuous Deployment (CI/CD) have grown in popularity in software development because they allow for a more streamlined and efficient software development process. Jenkins, an open-source automation server that aids in the automation of elements of the software development process, is a popular CI/CD platform. This project entails delivering a Node.js and JavaScript application on Azure via a CI/CD pipeline with Jenkins. The application code is kept in a GitHub repository, and a Docker container image is kept on Docker Hub. The CI/CD pipeline will oversee automating the application's build, and deployment processes. When a new commit is made to the GitHub repository, the pipeline will be started automatically. After that, the pipeline will create the application, and publish it to Azure. The Docker container image published on Docker Hub will be used to generate a new container on an Azure virtual machine during the deployment process. The pipeline will also build up the virtual machine with the necessary environment variables and settings for the program to operate. Overall, using a Jenkins and Azure CI/CD pipeline will assist in streamlining the software development process, eliminate mistakes and defects, and ensure that the application is released fast and efficiently. The conversion procedure was quite efficient, demonstrating the database's perfect integration into the AWS server environment. This smooth transition is the result of careful preparation and execution during the relocation process. Stringent verification techniques were used to assure data consistency and integrity after transfer, which is crucial for the correctness and dependability of the customized database within the AWS architecture. This demonstrates our dedication to a smooth and reliable database transfer, providing the groundwork for maximum performance in the AWS architecture.

**Source**

The first step in starting the CI/CD pipeline is to set up a code repository and version control. We set up a GitHub repository, which will be useful for managing the code. Team members may simply push changes to a shared repository using Git, and any team member can get updated code from others using pull. The pipeline can be set to run automatically or manually started by the user. We will use the JavaScript programming language to write the source code, and Node JS for front-end UI implementation.

**Source Code Git Hub Url :** <https://github.com/jyothimaria/Devops_project.git>

**Branches: master**

**Web Development and Database Integration:**

We successfully designed and built a web application during the local system development phase. The establishment of a secure login page, with login credentials kept systematically in a MySQL database, is a critical component of this program. The development process was fueled using the JavaScript programming language, which was used to create a dynamic and responsive application. Furthermore, Node.js played an important part in front-end development, improving the user interface for an ideal user experience. This comprehensive strategy aims to discover and fix any potential difficulties, ensuring the web application's seamless and dependable execution.

A screenshot of a computer

Description automatically generated*figure 2: Login Page*

A screenshot of a computer

Description automatically generated*figure 3: After login, it will be redirected to this page.*

A screenshot of a computer

Description automatically generated*figure 4: Backend Database*

**Build**

With a completely working web application source code posted on GitHub, our next step was to create a containerized version that was ready for deployment. Here successfully encased the application using Docker, assuring consistency, reliability, and portability. The Dockerized application was then deployed on an Azure Virtual Machine (VM), utilizing Azure's powerful infrastructure to successfully host and manage the application and used SQL as the database to store and maintain data, guaranteeing a safe and scalable data management solution for our online application.

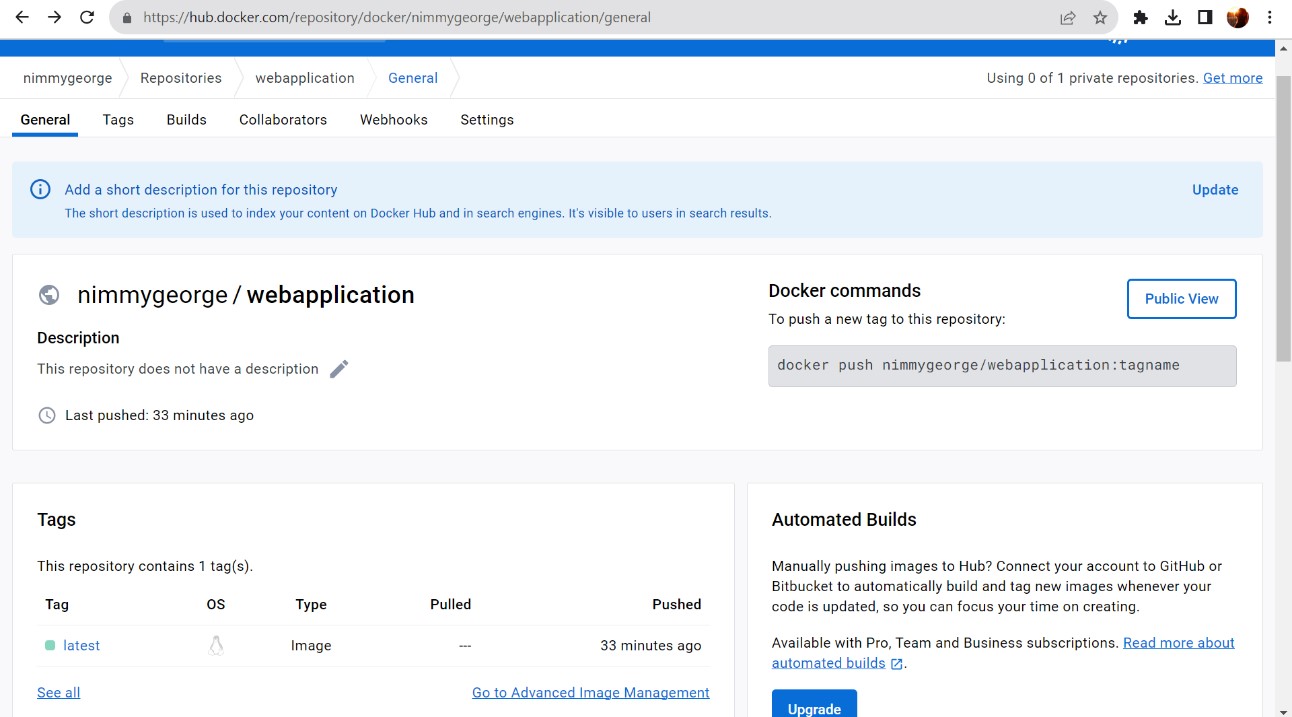
During this critical step of the software development lifecycle, the source code packages are meticulously rebuilt to ensure their capacity to generate artefacts for eventual deployment into the target environment. This cyclical approach entails the construction of a Docker image that encapsulates the application and its dependencies, as well as the generation of a well-organized artefact, which is frequently in the form of a container image. This artefact is a deployable item that wraps the program and its runtime environment to ensure consistency across environments. npm (Node Package Manager) functionalities are used to aid in the development of the source code. npm, a JavaScript package manager, is useful for managing dependencies, running build scripts, and managing project configurations. This methodical approach to source code creation and artefact development improves the software deployment process's efficiency, reliability, and consistency.

### **Application Dockerization**

The initiation of Docker Hub repository was a critical step in creating a centralized hub for storing our project's container images. This repository was painstakingly set up to comply with version control and access control criteria, creating a solid basis for the later phases of our containerization process. With the repository in place, we moved on to the Dockerization of our application, which went off without a hitch. This important step not only ensured consistency and reliability, but also granted our application the desirable attribute of portability across varied settings, which was done by enclosing the program in a Docker container.

A screenshot of a computer

Description automatically generated *figure 5: Docker Hub repository created.*

 *figure 6: Docker Hub repository general* ***Application***

Crossed a critical threshold when the containerized application successfully built a Docker container. This encapsulation enables us to deliver the program in a consistent and effective manner, regardless of the underlying infrastructure. As a result, the Docker container that included project application and all its dependencies was placed on the Docker Hub repository. Centralized hosting means that container images are kept and delivered from a single repository, hence improving our application's accessibility and distribution.

A screenshot of a computer program

Description automatically generated*figure 7: Build the application by using Docker.*

**Docker Container Creation:**

To improve version traceability and management, used a rigorous tagging and versioning strategy. The container image was manually tagged with version or release IDs. This purposeful tagging process improves version control and traceability by giving a clear and organized history of our application's progress. Furthermore, for greater flexibility and control, we adopted the practice of establishing several tags to represent different versions or releases. This sophisticated tagging method not only supports effective version management but also accommodates a wide range of use scenarios, indicating a purposeful effort to improving the overall maintainability of our containerized program.

A screenshot of a computer program

Description automatically generated *figure 8: Docker container created.*

### **Docker Hub Image Push**

**Centralized Hosting:** The Docker container successfully published the project application and all its dependencies to the Docker Hub repository. This repository currently stores and distributes container images.

**Tagging and Version Control:**

Traceability of Versions: The container image was manually marked with a specific version or versions.

To facilitate version control, release IDs are used. Traceability and version control

This technique simplifies everything.

**Tagging Variants**:

Several tags may have been generated to indicate different versions or releases to increase flexibility and administration.

**Deploy**

### **Azure Virtual Machine Creation**

Development of an Azure Virtual Machine (VM), which was a major milestone. The virtual machine (VM) was carefully set up with the necessary specs and resources to host the project application. The installation process for Docker on the Azure virtual machine was executed successfully, marking a pivotal milestone in our deployment strategy. This installation is of paramount importance as it enables the containerization of our application, providing a standardized deployment environment that enhances consistency and portability across various contexts. The utilization of Docker ensures that our application and its dependencies are encapsulated within containers, facilitating efficient deployment and scalability. Additionally, Jenkins, a powerful automation tool, was successfully installed on the Azure virtual machine to establish the foundation for our Continuous Integration/Continuous Deployment (CI/CD) pipeline. This strategic combination of Docker and Jenkins sets the stage for an automated and streamlined development workflow, reinforcing our commitment to efficient, reliable, and scalable application deployment in the Azure environment.

A screenshot of a computer

Description automatically generated *figure 9: Azure VM configured with port number.*

A screenshot of a computer

Description automatically generated*figure 10: Azure VM connected by local PowerShell.*

A screenshot of a computer

Description automatically generated*figure 11: Jenkins installed and status-Active.*

### **CI/CD Pipeline**

The deployment phase is at the center of the CI/CD (Continuous Integration/Continuous Deployment) pipeline, which aims to simplify and automate the process from development to production. To accomplish this goal, the solution entails building pipeline tasks in Jenkins, a popular automation server. These tasks are precisely built to handle various components of the deployment process, including application development, and packaging. The incorporation of these activities into the Jenkins pipeline creates a unified flow, ensuring that each step is completed effectively and without the need for manual intervention. To improve deployment capabilities, the CI/CD pipeline makes use of Jenkins plugins designed expressly for continuous integration. These plugins are critical in extending Jenkins' capability, allowing it to smoothly interface with numerous tools and technologies, handle dependencies, and conduct various activities in the deployment pipeline. Furthermore, Azure, Microsoft's cloud computing platform, is used to optimize the deployment process. Azure delivers a strong and scalable architecture that not only allows for rapid deployment but also capabilities for monitoring, scaling, and managing cloud-based applications. Organizations may establish a dependable and automated deployment pipeline by integrating Jenkins' adaptability with Azure's capabilities, resulting in a more agile and responsive software delivery lifecycle.

A screenshot of a computer screen

Description automatically generated *figure 12: Docker deployment*

A screenshot of a computer

Description automatically generated

*figure 13: Jenkins installed and status-Active.*

A screenshot of a computer

Description automatically generated *figure 14: Jenkins Started.*

A screenshot of a computer

Description automatically generated *figure 15: Jenkins Account setup*

A screenshot of a computer

Description automatically generated *figure 16: Docker and GitHub credentials configured.*

A screenshot of a computer

Description automatically generated *figure 17: Pipeline Setup*

**Database Migration**

Here used the AWS Database Migration Service to transfer our SQL database to Amazon Web Services (AWS) as the final stage in our deployment plan. This strategic decision offers us with a scalable and high-performance database solution on AWS, which aligns with our long-term scalability and performance objectives.

### **AWS Server Integration**

The conversion procedure was quite efficient, demonstrating the database's flawless integration into the AWS server environment. This smooth transition highlights the painstaking preparation and execution that goes into the migration procedure. To ensure data consistency and integrity after migration, stringent verification processes were meticulously followed. These steps were critical in ensuring that the migrated database acclimated to the AWS server environment without jeopardizing the correctness or dependability of the stored data. The emphasis on integration efficacy as well as rigorous adherence to verification processes demonstrates our dedication to guaranteeing a seamless and reliable database conversion, creating the groundwork for maximum performance inside the AWS architecture.

**mysqldump to export.**

The mysqldump program was critical in efficiently exporting data from our MySQL database. We were able to build a textual representation of the database's structure and contents using this command-line program. The procedure entailed making a logical backup that included SQL statements capable of reproducing the complete database when performed. This strategy not only simplifies data movement but also offers a dependable backup for recovery and replication. The use of mysqldump ensured a simple and standardized method to data exporting, providing flexibility and convenience of use in controlling the structure and content of the database. We shortened the export procedure by carefully employing mysqldump, contributing to the overall efficiency and dependability of our database administration practices.

A screenshot of a computer program

Description automatically generated*figure 18: SQL dump*

**Created rds instance with name.**

The setting up of an Amazon RDS (Relational Database Service) instance was a critical step in developing a scalable and controlled database solution for our application. The RDS instance was carefully set up and launched with a unique name, which will serve as an identification inside our cloud infrastructure. This managed database service gave us the freedom to pick alternative database engines, automated backups, and seamless scalability based on our application's increasing demands. We were able to focus on the essential components of our application development since we used Amazon RDS to offload the administrative responsibilities associated with database administration. The named RDS instance now serves as a dedicated and trustworthy basis for storing and retrieving data, completely matching with Amazon's scalability and ease-of-management benefits.

A screenshot of a computer

Description automatically generated*figure 19: AWS RDS*

**Used mysql command to migrate the instance.**

The MySQL command was used to rapidly conduct the migration procedure, which was a critical step in successfully migrating our database instance. We started a methodical transfer of data, schema, and configurations to the specified database instance using the MySQL command-line interface. This command made it easier to synchronize our MySQL database from the source to the target instance, assuring constant and reliable data replication. Database credentials, source, and destination were precisely set to meet our migration needs. The MySQL command allowed us to oversee the migration process with precision and accuracy since it gave a direct and controlled method. This migration not only protected data integrity, but also allowed us to transfer our database seamlessly, assuring its accessibility and stability inside the new instance setup.

A screenshot of a computer

Description automatically generated*figure20: Database configuration using AWS.*

A screenshot of a computer

Description automatically generated*figure 21: Application running with AWS database*

**Results**

A screenshot of a computer

Description automatically generated*figure 22: Application hosted on Azure VM*

A screenshot of a computer

Description automatically generated

*figure 23: pipeline success*

A screenshot of a computer

Description automatically generated

*figure24: console output*

A screenshot of a computer

Description automatically generated

*figure25: Database Migration successfully*

# **Conclusions and Future Work**

In conclusion, this project successfully passed three important stages, beginning with the construction of a feature-rich online application utilizing JavaScript and Node.js, which was hosted on GitHub for efficient version control. The construction process included Docker containerization and deployment on an Azure Virtual Machine using SQL as the database of choice. A CI/CD pipeline was used to simplify automation, assuring dependable updates. The project resulted in scalability optimization by transferring the SQL database to AWS using the Database Migration Service. This holistic strategy represents the effective integration of current development practices, cloud infrastructure, and collaboration tools, resulting in a durable and scalable web application capable of delivering efficient performance in a variety of situations.

# **References**

*Tutorial: Create a jenkins pipeline using GitHub and Docker*. GitHub. (n.d.). [https://github.com/MicrosoftDocs/azure-dev-docs/blob/main/articles/jenkins/pipeline-with-github-and-docker.md](https://github.com/MicrosoftDocs/azure-dev-docs/blob/main/articles/jenkins/pipeline-with-github-and-docker.md%20)

Burns, S. (2022, May 23). *How to run Docker on an azure VM: TechTarget*. IT Operations. <https://www.techtarget.com/searchitoperations/tip/How-to-run-Docker-on-an-Azure-VM>

*Manage repositories*. Docker Documentation. (2023a, August 22). [https://docs.docker.com/docker-hub/repos/](https://docs.docker.com/docker-hub/repos/%20)

Det Danske Missionsselskab. (1989a). *DMS: Aktuelle Retningslinier*. Amazon. <https://aws.amazon.com/dms/>

GeeksforGeeks. (2023, April 13). *How to make a CI-CD pipeline in Jenkins?*. GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-make-a-ci-cd-pipeline-in-jenkins/>

Khan, H. (n.d.). *Setting up your CI/CD pipeline: Jenkins and github: Blazemeter by perforce*. BlazeMeter. <https://www.blazemeter.com/blog/cicd-pipeline-jenkins-github>

Team, C. L. D. (n.d.). *Migrate with AWS Database Migration Service (DMS)*. CockroachDB Docs. <https://www.cockroachlabs.com/docs/stable/aws-dms>