

# Application DevOps Pipeline with Jenkins CI/CD and AKS Containerization

---

## Table of Contents

### Introduction

- Overview of DevOps and CI/CD
- Role of Jenkins in Continuous Integration and Continuous Deployment (CI/CD)
- Importance of Kubernetes and AKS (Azure Kubernetes Service) in containerized applications
- Project objectives and expected outcomes

### Literature Review

- Concepts of DevOps and its evolution
- Overview of CI/CD principles and tools
- Jenkins as a CI/CD tool and its integration capabilities
- Containerization and orchestration with Kubernetes
- Azure Kubernetes Service (AKS) and its advantages for deploying scalable applications

### Project Scope and Objectives

- Scope of the DevOps pipeline with Jenkins and AKS
- Objectives of the project:
  - Automating build, test, and deployment stages
  - Ensuring scalability and reliability of the application
  - Securing deployment on AKS
- Project constraints and assumptions

### Technology Stack

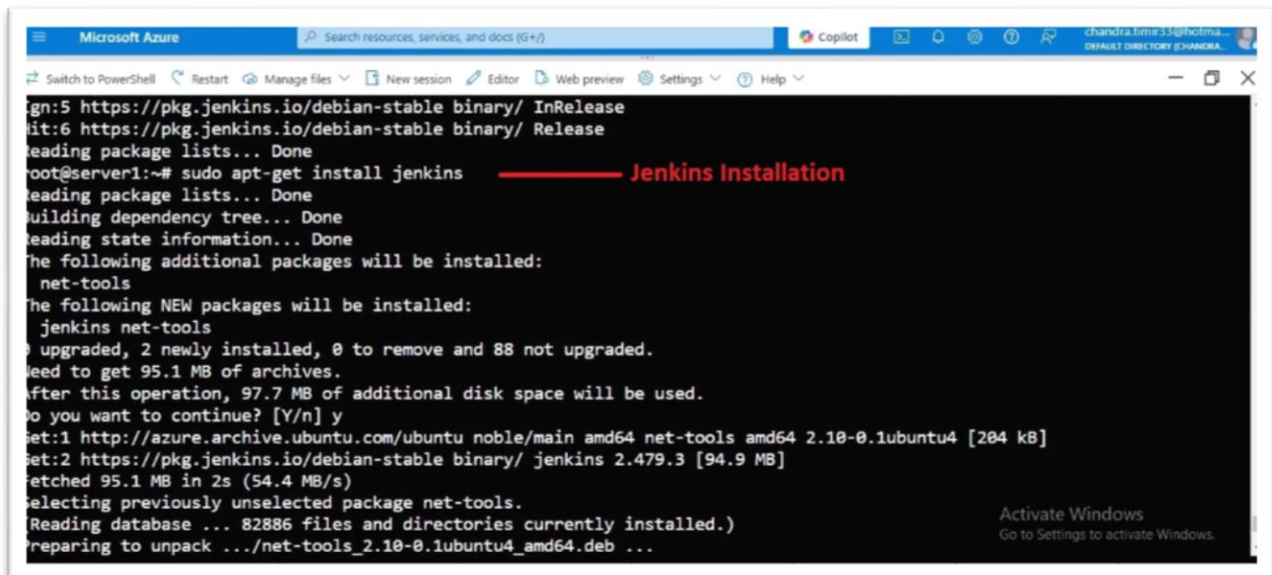
- Jenkins: Overview, setup, and plugins
- Docker: Containerization of applications
- AKS (Azure Kubernetes Service): Features, setup, and integration with Jenkins
- GitHub/GitLab: Code repository and version control
- Other tools (optional): Helm for Kubernetes package management, Terraform for infrastructure as code, etc.

### Pipeline Design and Architecture

- High-level architecture of the CI/CD pipeline
- Stages of the pipeline:
  - Code commit and version control with GitHub/GitLab
  - Build and testing stage using Jenkins
  - Containerization with Docker
  - Deployment to AKS
- Detailed workflow of each stage with diagrams
- Monitoring and logging setup for pipeline feedback

## Pipeline Implementation

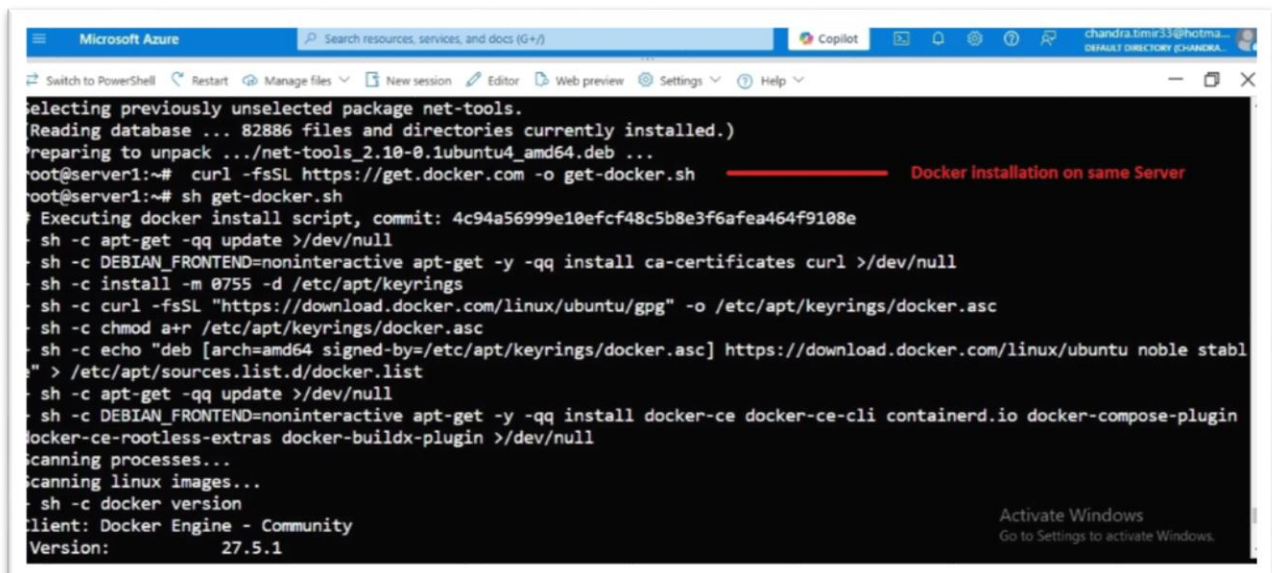
- Step-by-step setup and configuration:
  - Install Jenkins and Docker on the same server



```

gn:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
git:6 https://pkg.jenkins.io/debian-stable binary/ Release
Reading package lists... Done
root@server1:~# sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 88 not upgraded.
Need to get 95.1 MB of archives.
After this operation, 97.7 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
get:1 http://azure.archive.ubuntu.com/ubuntu noble/main amd64 net-tools amd64 2.10-0.1ubuntu4 [284 kB]
get:2 https://pkg.jenkins.io/debian-stable binary/ jenkins 2.479.3 [94.9 MB]
Fetched 95.1 MB in 2s (54.4 MB/s)
Selecting previously unselected package net-tools.
(Reading database ... 82886 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
  
```

Activate Windows  
Go to Settings to activate Windows.

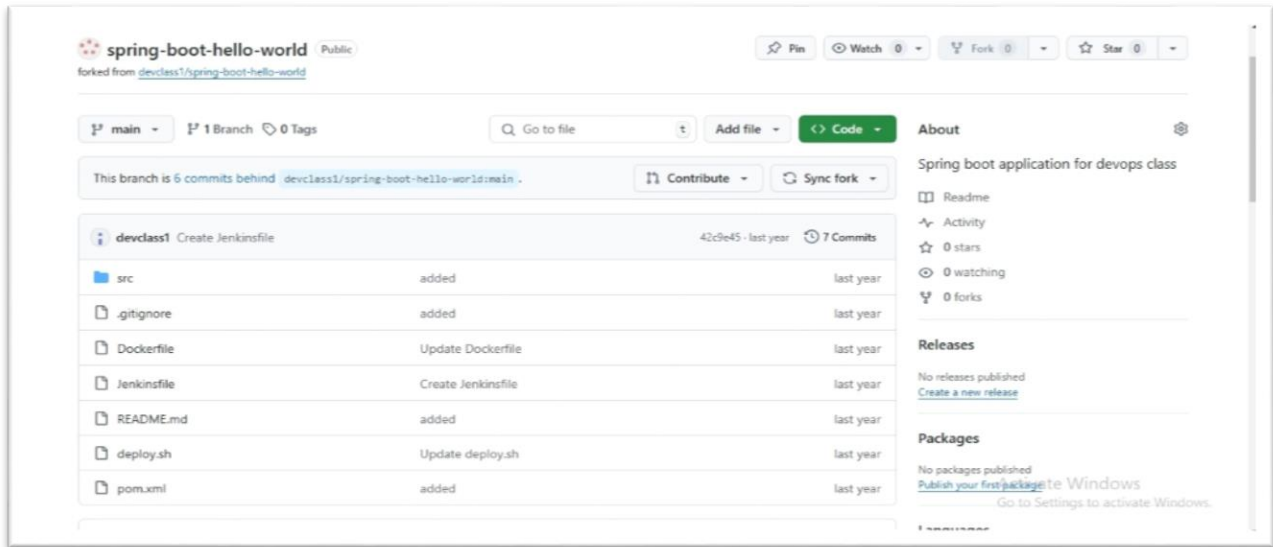


```

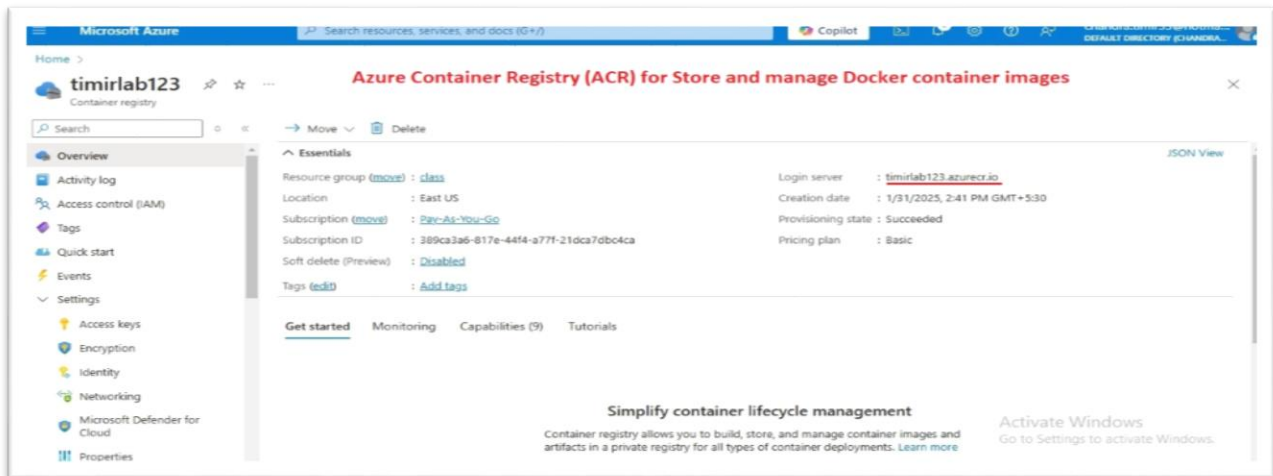
Selecting previously unselected package net-tools.
(Reading database ... 82886 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
root@server1:~# curl -fsSL https://get.docker.com -o get-docker.sh
root@server1:~# sh get-docker.sh
Executing docker install script, commit: 4c94a56999e10efcf48c5b8e3f6afea464f9108e
sh -c apt-get -qq update >/dev/null
sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install ca-certificates curl >/dev/null
sh -c install -m 0755 -d /etc/apt/keyrings
sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" -o /etc/apt/keyrings/docker.asc
sh -c chmod a+r /etc/apt/keyrings/docker.asc
sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu noble stable" > /etc/apt/sources.list.d/docker.list
sh -c apt-get -qq update >/dev/null
sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install docker-ce docker-ce-cli containerd.io docker-compose-plugin
docker-ce-rootless-extras docker-buildx-plugin >/dev/null
Scanning processes...
Scanning linux images...
sh -c docker version
Client: Docker Engine - Community
Version: 27.5.1
  
```

Activate Windows  
Go to Settings to activate Windows.

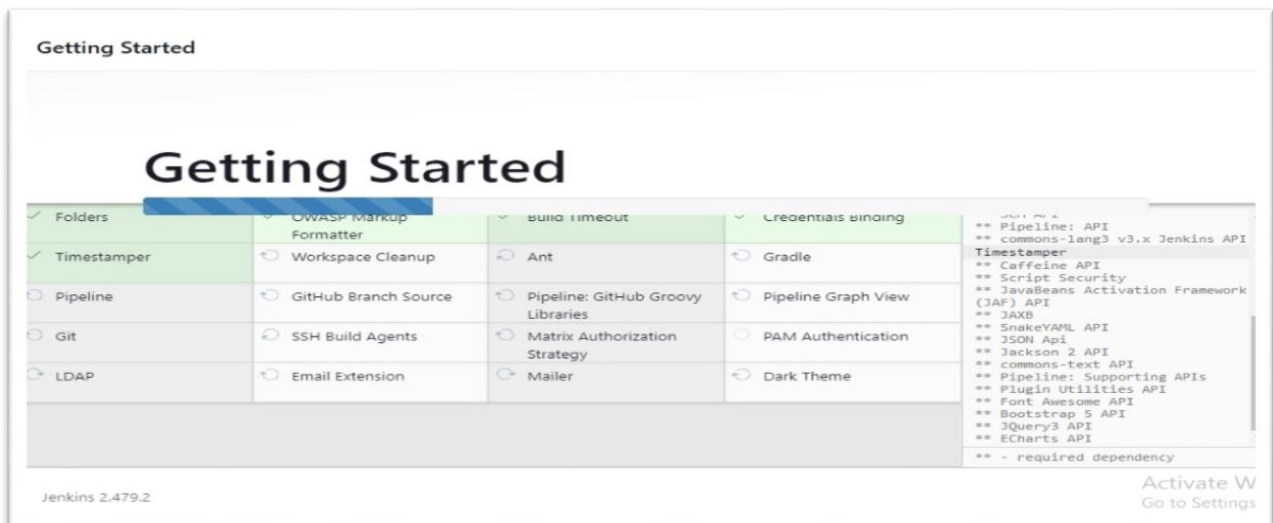
- Git Repository name as a **spring-boot-hello-world** with Dockerfile



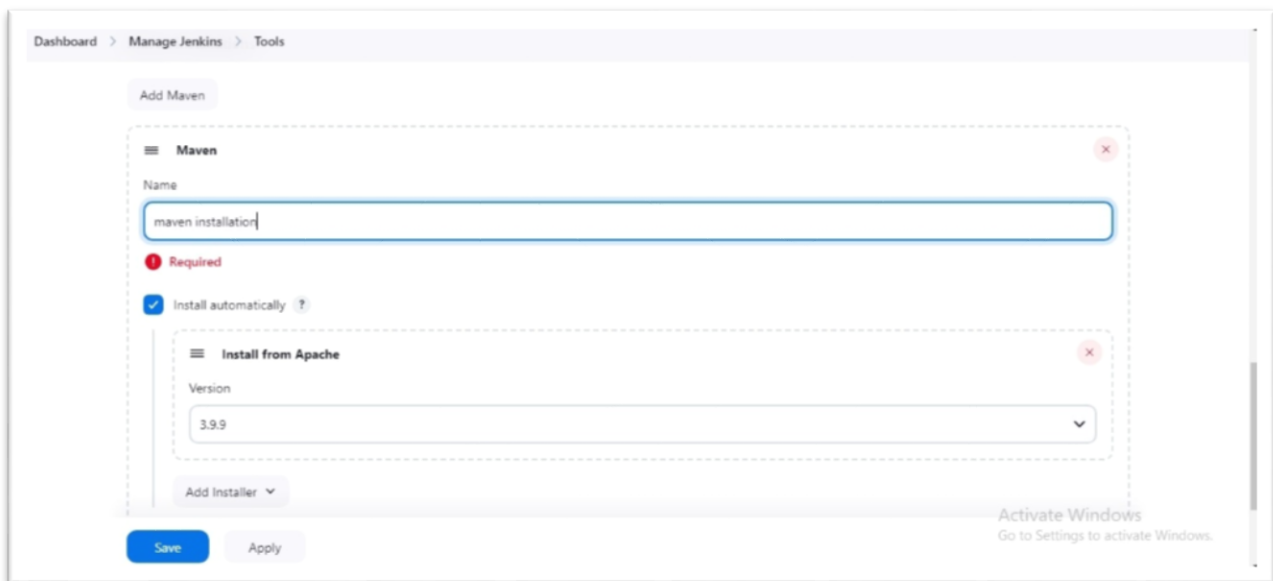
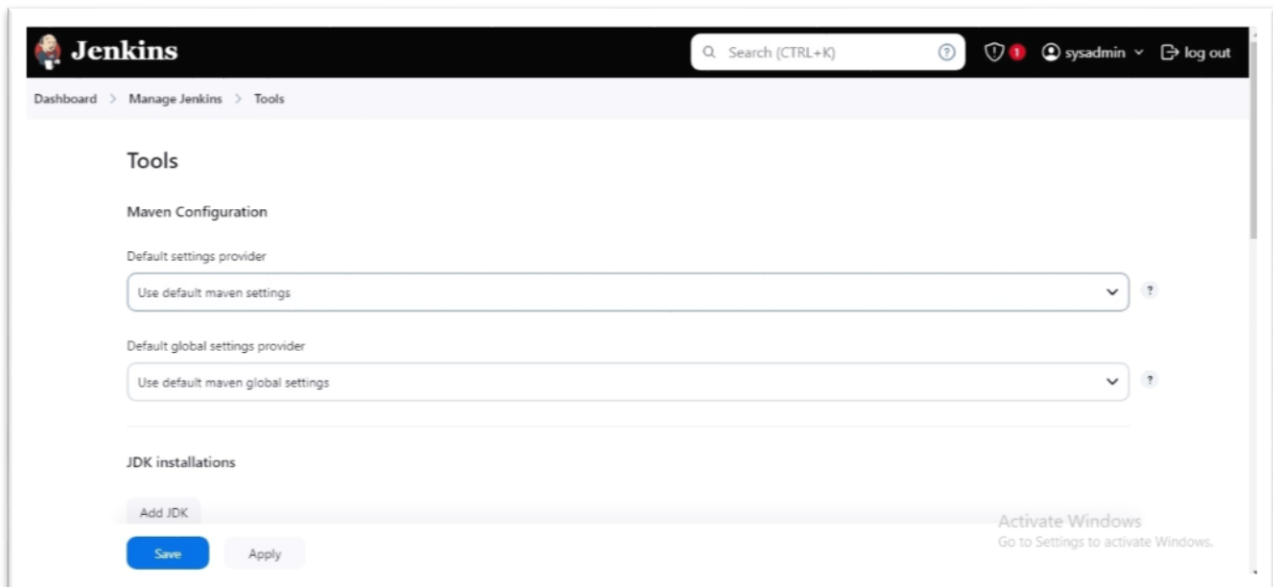
- **Azure Container Registry (ACR) for Store and manage Docker container images**



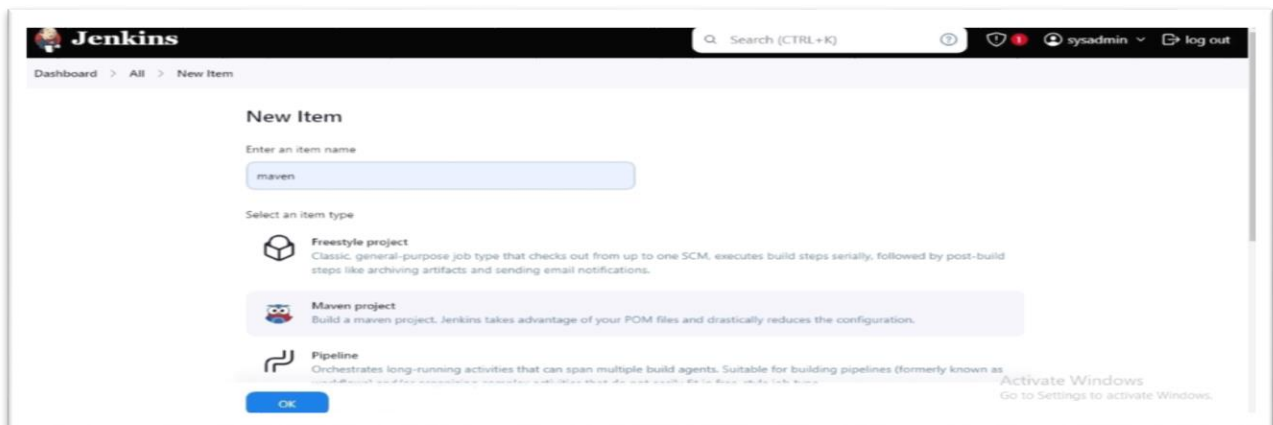
- Jenkins setup with relevant maven plugins & tools







- Creating a Spring Boot project with the name 'Maven'



- **Configuration General Setting**

Dashboard > maven > Configuration

### Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

### General

Enabled ☒

Description: Springboot Project

Plain text Preview

☐ Discard old builds ?

☒ GitHub project

Project url ? https://github.com/timrchandra/spring-boot-hello-world

Save Apply

Activate Windows  
Go to Settings to activate Windows.

- **Source Code Management**

Dashboard > maven > Configuration

### Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

### Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ? https://github.com/timrchandra/spring-boot-hello-world.git

Credentials ? - none -

+ Add

Save Apply

Activate Windows  
Go to Settings to activate Windows.

- **Build Triggers**

Dashboard > maven > Configuration

### Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

### Build Triggers

☒ Build whenever a SNAPSHOT dependency is built ?

☐ Schedule build when some upstream has no successful builds ?

☐ Trigger builds remotely (e.g., from scripts) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

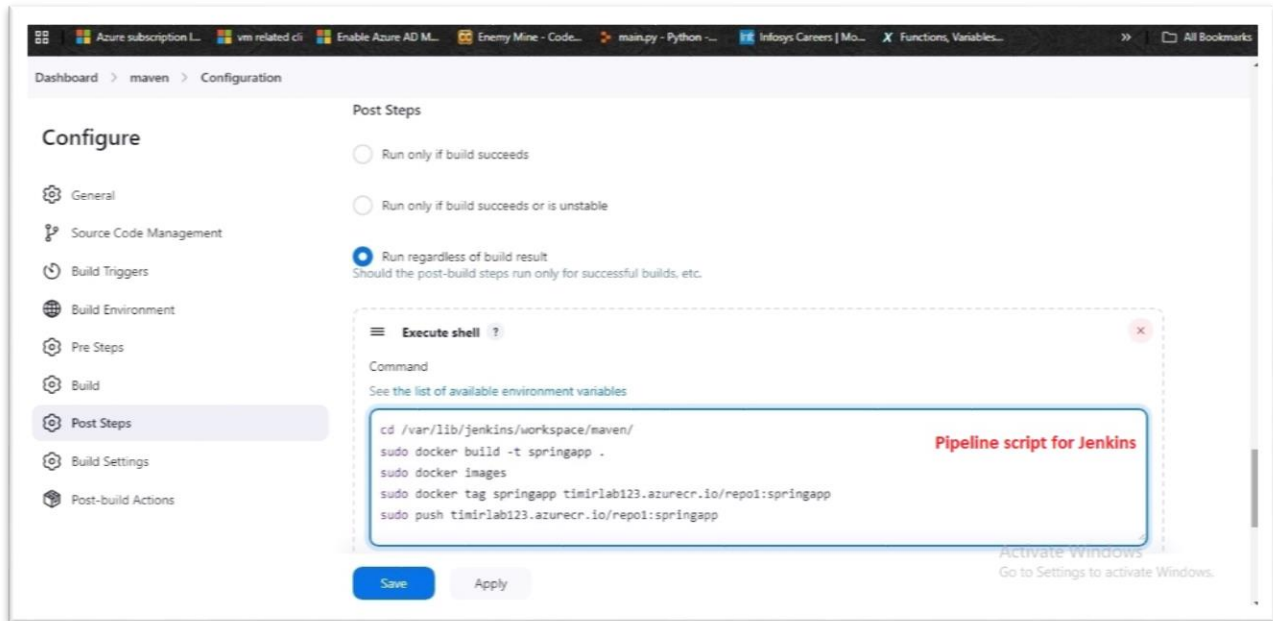
### Build Environment

☐ Delete workspace before build starts

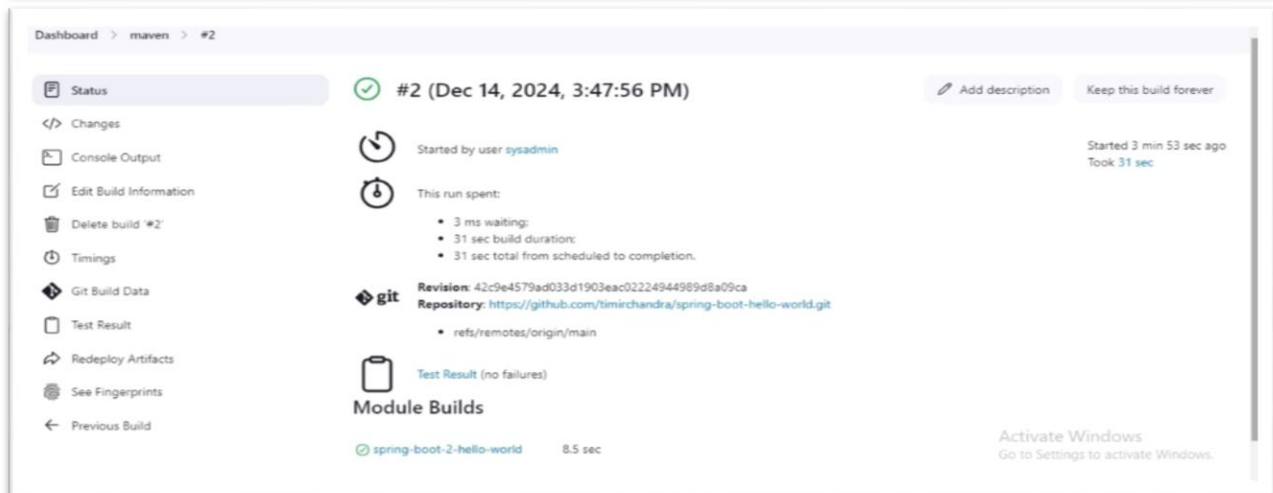
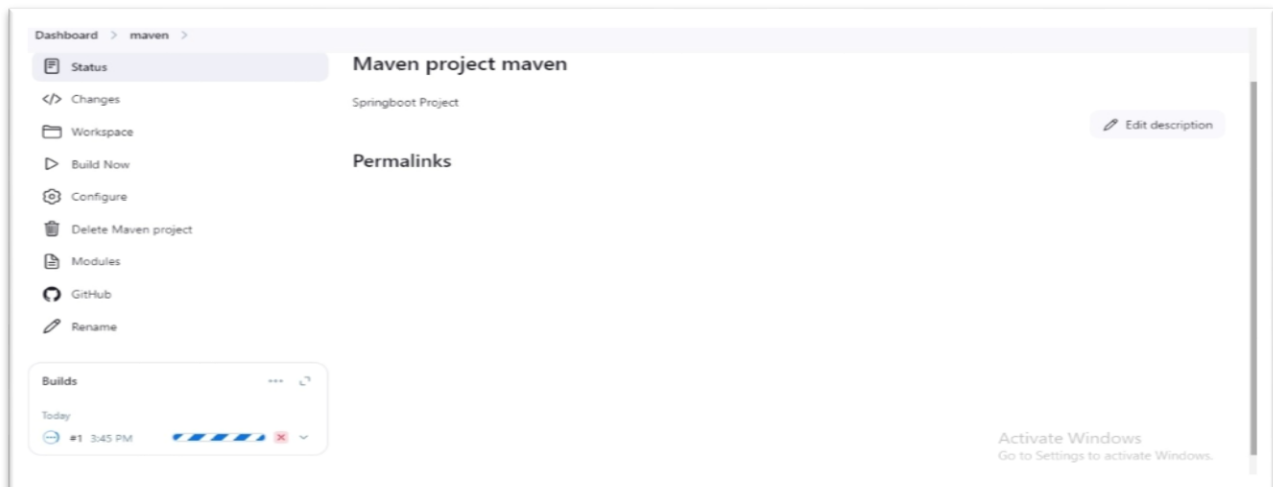
Save Apply

Activate Windows  
Go to Settings to activate Windows.

- Pipeline as Code: Jenkinsfile setup for CI/CD stages
- Write pipeline script for Jenkins, then save and apply it

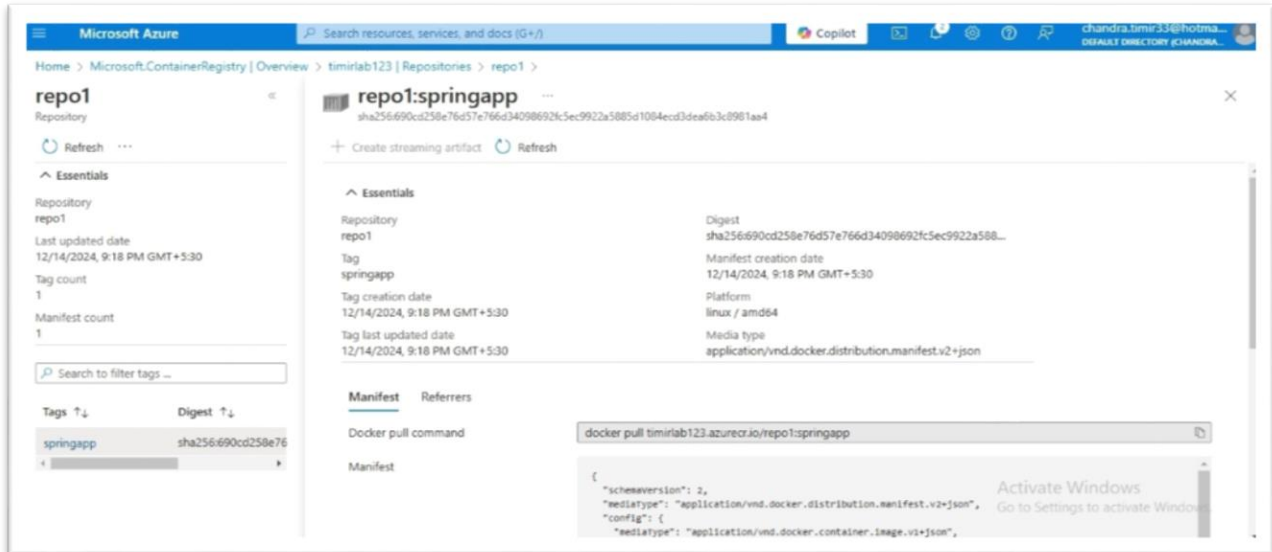


- Then trigger the build

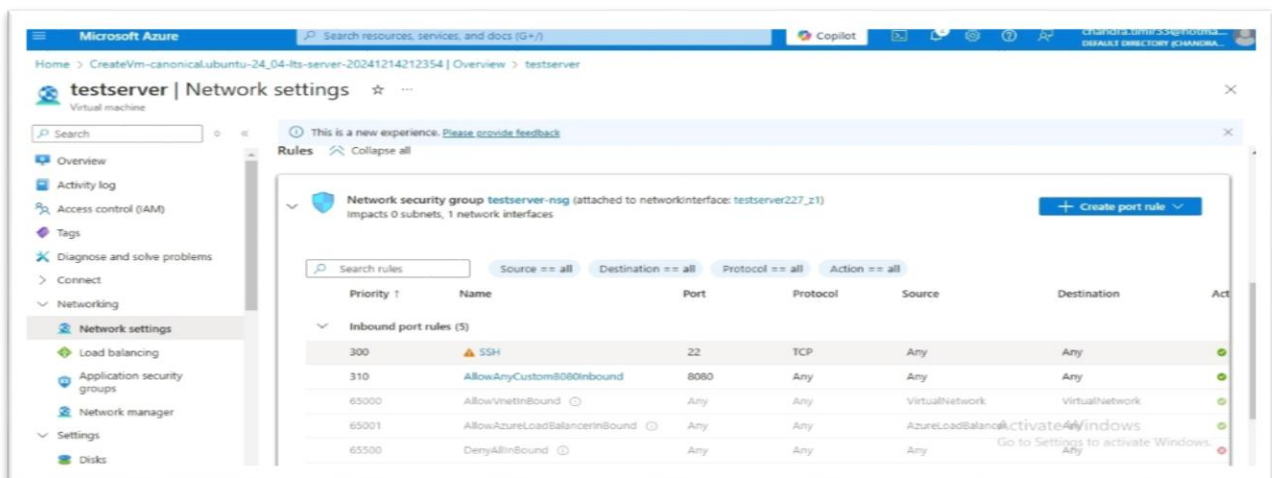




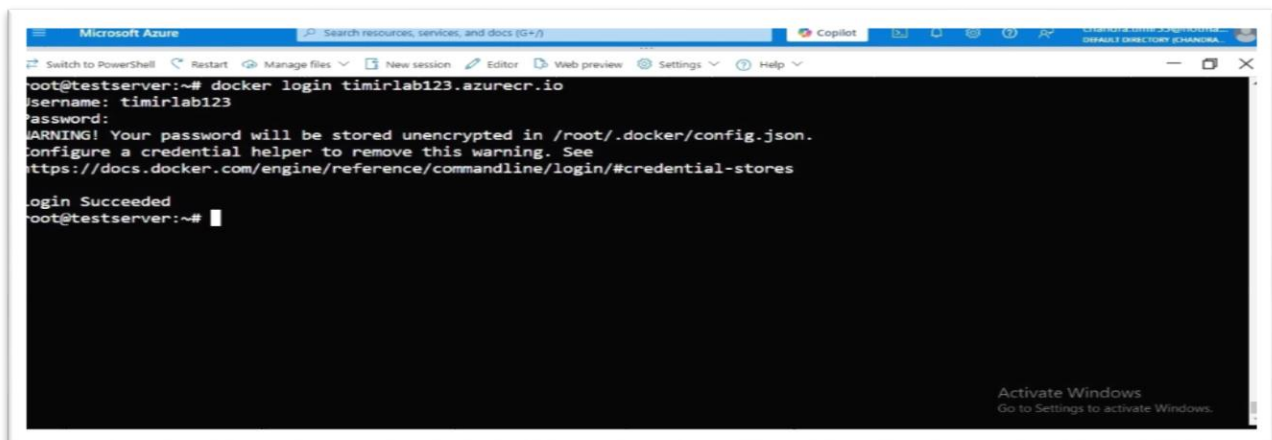
- After successfully running the script, the custom image is pushed to the Azure Container Registry (ACR)



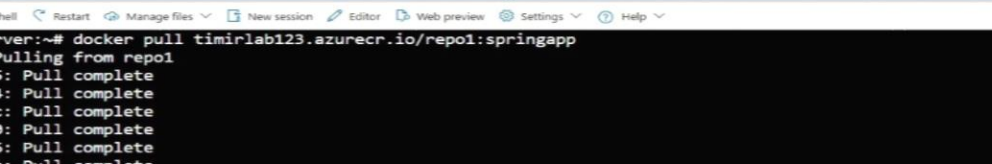
- After successfully pushing the image to ACR, we need to test the custom image to ensure it was built properly.
- For testing purposes, we have launched a test server where Docker is already installed and running on port 8080



- Attach the test server to ACR (timirlab123) for image pull







The screenshot shows a Windows terminal window with the title bar "Microsoft Azure". The terminal content is as follows:

```
root@testserver:~# docker pull timirlab123.azurecr.io/repo1:springapp
springapp: Pulling from repo1
001c52e26ad5: Pull complete
09d4b9b6e964: Pull complete
068746827ec: Pull complete
0daef329d350: Pull complete
085151f15b66: Pull complete
02a8c426d30b: Pull complete
0754a66e0050: Pull complete
09fcf55a1eb64: Pull complete
Digest: sha256:690cd258e76d57e766d34098692fc5ec9922a5885d1084ecd3dea6b3c8981aa4
Status: Downloaded newer image for timirlab123.azurecr.io/repo1:springapp
timirlab123.azurecr.io/repo1:springapp
root@testserver:~#
```

An "Activate Windows" watermark is visible in the bottom right corner of the terminal window.

[illegible]

A screenshot of a web browser window. The address bar shows a URL starting with 'http://localhost:5000/'. The page title is 'Whitelabel Error Page'. The main content area displays the message: 'This application has no explicit mapping for /error, so you are seeing this as a fallback. Sat Dec 14 16:07:18 UTC 2024 There was an unexpected error (type=Not Found, status=404).' The browser's taskbar at the bottom shows several open applications, including 'Azure subscription L...', 'vm related cli', 'Enable Azure AD M...', 'Enemy Mine - Code...', 'main.py - Python ~...', 'InfoSys Careers | Mo...', 'Functions, Variables...', and 'All Bookmarks'.

- After successfully passing the testing, we will proceed to the production phase. For this purpose, we need to launch AKS (e.g., the cluster name is mycluster).

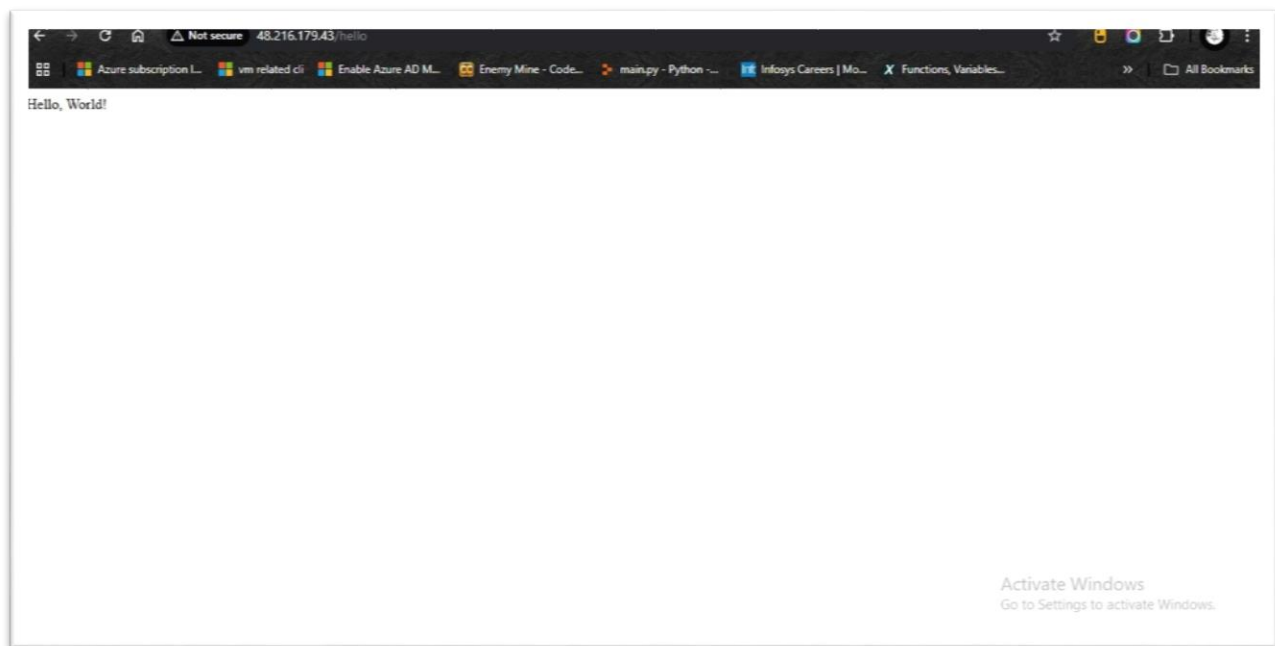
The screenshot shows the Microsoft Azure portal interface. On the left, the 'mycluster' Kubernetes service overview is displayed, including details like Resource group (class), Power state (Running), and Location (East US). On the right, a sidebar titled 'Connect to mycluster' provides instructions for connecting to the cluster using the Cloud shell, Azure CLI, or Run command. The 'Cloud shell' tab is selected, showing steps to open the shell and run commands to set the cluster context and download credentials.

The screenshot shows the Azure Cloud Shell terminal. It displays the successful connection to the Cloud Shell and the execution of several commands to attach the Azure Container Registry (ACR) to the 'mycluster' Kubernetes cluster. The commands include setting the subscription, getting credentials, and updating the cluster configuration. A red text overlay at the bottom reads 'Attach ACR to mycluster'.

- YAML file for application deployment.

The screenshot shows the Azure Cloud Shell terminal displaying a YAML file for application deployment. The file is named 'springapp.yml' and contains configuration for a deployment named 'helloworld-deployment'. It specifies the image as 'timirlab123.azurecr.io/repol:springapp' and sets the container port to 8080. The deployment is configured with 2 replicas and a selector matching the 'app: helloworld' label.





## Security and Compliance

- Security considerations in the DevOps pipeline
- Securing Jenkins with authentication and access control
- Container security best practices (e.g., image scanning)
- Kubernetes security and AKS-specific security practices
- Compliance with best practices and industry standards

## Testing and Quality Assurance

- Overview of testing strategy for CI/CD
- Unit, integration, and functional testing
- Performance and load testing on AKS

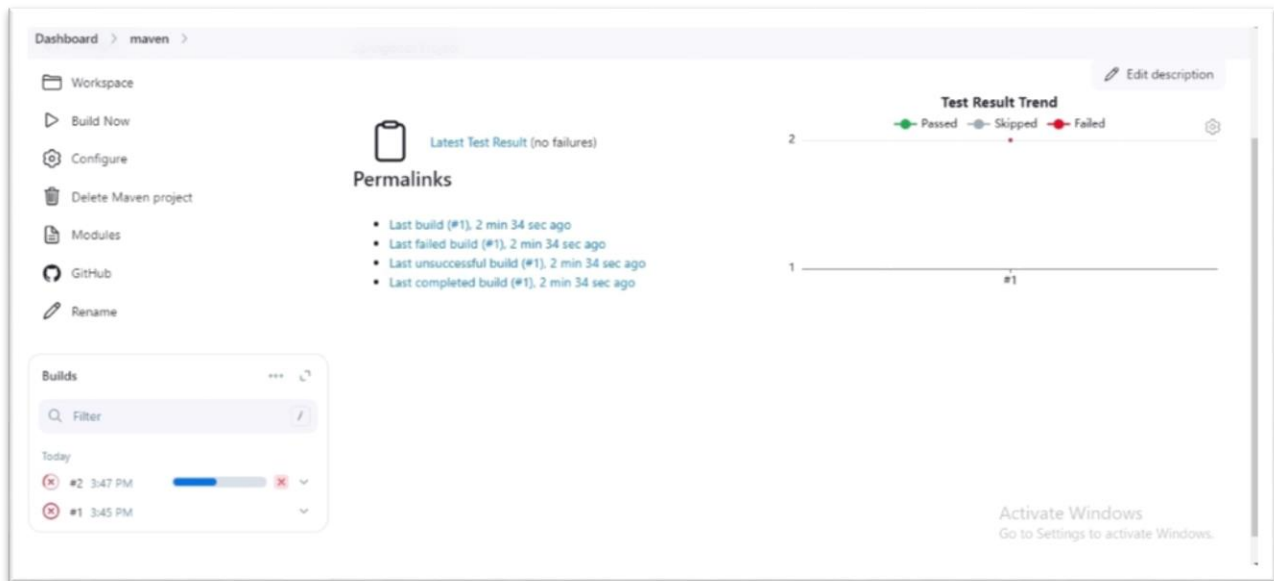
## Monitoring and Logging

- Monitoring tools and techniques for the Jenkins pipeline
- Application performance monitoring on AKS
- Logging for error tracking and troubleshooting
- Integrating monitoring with Jenkins and AKS

## Challenges and Solutions

- Technical and procedural challenges in setting up the pipeline failed during the image build or push stages due to incorrect configurations, missing dependencies.

- Authentication with the Azure Container Registry (ACR) failed, causing the push to the registry to be rejected.
- **Solutions implemented to overcome challenges:** After reviewing the error logs, we identified the specific stage where the pipeline was failing and focused on resolving the issue step by step. We addressed issues related to missing files or incorrect paths in the `Dockerfile` and added the necessary files and dependencies to the Docker image build context
- **Lessons learned during implementation :**
  - **Importance of Detailed Error Logs:** Always analyze error messages in detail. Often, the error logs provide direct hints about the root cause of the issue (e.g., authentication failures, missing files, incorrect paths)



- **Check Dockerfile Context:** Verify the `Dockerfile` and project directory structure. Any missing dependencies or incorrect paths in the build context can cause unexpected failures.

## Results and Analysis

- Analysis of the outcomes from the pipeline
- Benefits of the automated pipeline for development and deployment
- Performance metrics and improvements observed with containerization and AKS

## Conclusion

- Summary of findings and project outcomes
- Future recommendations for enhancing the pipeline
- Final thoughts on the role of DevOps, CI/CD, and AKS in modern application development

## References

- List of references for tools, techniques, and theories discussed in the report

## Appendices

- Additional diagrams, code snippets, and configuration files

---

# Introduction

## ● Overview of DevOps and CI/CD

DevOps is a set of practices that integrates software development (Dev) and IT operations (Ops) to enhance collaboration, automation, and efficiency throughout the software lifecycle. The goal is to deliver high-quality software quickly and reliably. Continuous Integration (CI) involves automatically integrating code changes into a shared repository, where tests are run to identify issues early. Continuous Deployment (CD) automates the process of deploying code to production after successful testing, ensuring fast and reliable delivery of features. Together, CI/CD streamline workflows, reduce errors, and enable rapid, consistent delivery of applications with improved collaboration and faster time to market.



- Role of Jenkins in Continuous Integration and Continuous Deployment (CI/CD)

Jenkins plays a crucial role in Continuous Integration (CI) and Continuous Deployment (CD) by automating the build, test, and deployment processes in the software development lifecycle. In CI, Jenkins facilitates the integration of code from multiple developers into a shared repository by running automated tests on each commit, ensuring that errors are detected early. This reduces integration problems and improves collaboration among teams.

For CD, Jenkins automates the deployment of applications to different environments (e.g., staging, production) after successful testing. By integrating with various plugins, Jenkins can trigger deployments to cloud platforms like Azure, AWS, or Kubernetes environments, making the process seamless and reliable. The pipeline can include stages like code quality checks, unit tests, and deployment, ensuring that the application is always in a deployable state. Jenkins thus accelerates software delivery while maintaining quality, reducing manual intervention, and improving overall efficiency in DevOps practices.

- Importance of Kubernetes and AKS (Azure Kubernetes Service) in containerized applications

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It helps in managing clusters of containers across multiple machines, ensuring high availability, load balancing, and efficient resource utilization. Kubernetes is crucial for large-scale, distributed applications, providing features like self-healing, auto-scaling, and rollbacks.

Azure Kubernetes Service (AKS) is a managed Kubernetes service provided by Microsoft Azure, simplifying the process of deploying, managing, and scaling containerized applications. AKS automates tasks such as patching, upgrading, and monitoring, reducing operational overhead. It seamlessly integrates with Azure's ecosystem, offering enhanced security, scalability, and cost management. AKS enables businesses to leverage the power of Kubernetes while focusing on application development rather than infrastructure management, making it a critical tool for cloud-native applications.

- Project objectives and expected outcomes

The primary objective of this project is to design and implement a robust DevOps pipeline using Jenkins for Continuous Integration (CI) and Continuous Deployment (CD), coupled with Azure Kubernetes Service (AKS) for container orchestration. The pipeline will automate the entire software delivery process, ensuring faster, reliable, and scalable deployment of applications.

**Key objectives include:**

1. **CI/CD Automation:** Automate the build, test, and deployment processes to streamline the software development lifecycle and reduce manual intervention. This involves integrating Jenkins with Git repositories and configuring pipelines for automatic code integration, testing, and deployment.

2. **Containerization:** Use Docker to containerize applications, making them portable across different environments while ensuring consistency and scalability.
3. **Deployment on AKS:** Set up an Azure Kubernetes Service cluster for container orchestration, ensuring efficient scaling, load balancing, and high availability of applications in production environments.
4. **Security and Monitoring:** Implement security best practices and monitoring tools to ensure the safety, stability, and performance of the application during deployment and scaling.

Expected outcomes of the project include a fully automated CI/CD pipeline that reduces development cycles, enhances code quality through continuous testing, and ensures consistent, fast, and secure deployment of applications on scalable cloud infrastructure. This will significantly improve the speed and reliability of application releases.

---

## Literature Review

- Concepts of DevOps and its evolution

DevOps is a set of practices that integrates software development (Dev) and IT operations (Ops) to improve collaboration, automation, and efficiency throughout the software delivery lifecycle. It aims to shorten development cycles, increase deployment frequency, and ensure higher-quality software. The core principles of DevOps include automation, continuous integration (CI), continuous delivery (CD), collaboration, monitoring, and feedback loops.

### Evolution of DevOps:

- **Pre-DevOps (1990s):** Software development and IT operations were siloed, leading to slow deployments, miscommunication, and quality issues.
- **Agile Development (Early 2000s):** Agile methodologies emerged, emphasizing iterative development, fast feedback, and adaptability. However, the separation between development and operations remained.
- **DevOps Emergence (2007-2010):** The term "DevOps" was coined as a response to challenges faced by teams working in isolated silos. DevOps focuses on collaboration between development and operations teams, creating a culture of shared responsibility and continuous improvement.
- **DevOps Adoption (2010s):** The rise of cloud computing, automation tools, and containerization technologies (like Docker and Kubernetes) accelerated DevOps adoption. CI/CD pipelines, infrastructure as code (IaC), and microservices became fundamental to modern DevOps practices.

### Timeline:

1. **1990s:** Separate Dev and Ops roles.
2. **Early 2000s:** Agile principles emerge.
3. **2007-2010:** DevOps concept introduced.

#### 4. **2010s:** Widespread DevOps adoption with cloud and automation tools.

Today, DevOps continues to evolve with AI/ML integration, serverless architectures, and enhanced security measures.

- Overview of CI/CD principles and tools

Continuous Integration (CI) and Continuous Deployment (CD) are fundamental principles in modern software development, aiming to automate and streamline the process of building, testing, and deploying applications.

**Continuous Integration (CI)** involves frequently integrating code changes from multiple developers into a shared repository. This process triggers automated build and testing pipelines to ensure that any issues are identified and addressed early, enhancing collaboration and code quality.

**Continuous Deployment (CD)** extends CI by automating the deployment process. After passing automated tests, code is automatically deployed to production, ensuring that new features and updates are delivered rapidly and reliably. CD minimizes manual intervention and reduces the risk of human error in deployments.

Key **CI/CD tools** include:

2. **Jenkins:** A widely-used open-source automation server that orchestrates CI/CD pipelines.
3. **GitLab CI/CD:** A built-in tool for CI/CD automation in GitLab.
4. **CircleCI:** A cloud-based CI/CD tool focused on speed and efficiency.
5. **Travis CI:** A CI/CD service used with GitHub repositories.
6. **Azure DevOps:** A suite of tools for end-to-end DevOps automation.

- Jenkins as a CI/CD tool and its integration capabilities

Jenkins is a popular open-source automation tool for Continuous Integration (CI) and Continuous Deployment (CD) that facilitates the automation of building, testing, and deploying applications. It helps streamline the software development process by integrating code changes frequently and ensuring early detection of errors. Jenkins supports a wide range of plugins, allowing it to integrate with version control systems like Git, build tools such as Maven and Gradle, testing frameworks like JUnit, and deployment platforms like Kubernetes and cloud services.

Jenkins' integration capabilities extend to various DevOps tools, enabling end-to-end automation across the development lifecycle. It can trigger automated tests, initiate deployments, and send notifications, making it a central tool in DevOps pipelines for accelerating software delivery while maintaining high quality and consistency.

- Containerization and orchestration with Kubernetes

Containerization is the process of packaging an application and its dependencies into a lightweight, portable container that can run consistently across different environments. Docker is the most widely used containerization platform, providing isolation and scalability.

Kubernetes, an open-source container orchestration platform, manages the deployment, scaling, and operation of containerized applications. It automates tasks such as container scheduling, load balancing, and self-healing, ensuring high availability and reliability. Kubernetes supports features like automatic scaling, rolling updates, and service discovery, allowing organizations to efficiently manage large, complex applications in distributed environments, often in cloud-based infrastructures.

- Azure Kubernetes Service (AKS) and its advantages for deploying scalable applications

Azure Kubernetes Service (AKS) is a managed Kubernetes service provided by Microsoft Azure, designed to simplify the deployment, management, and scaling of containerized applications. AKS abstracts much of the complexity of Kubernetes management, such as cluster setup, patching, and maintenance, enabling developers to focus on building applications rather than managing infrastructure.

Key advantages of AKS include:

1. **Scalability:** AKS enables automatic scaling of containerized applications, adjusting resources based on demand.
2. **High Availability:** AKS ensures fault tolerance by distributing containers across multiple nodes and availability zones.
3. **Integrated Azure Services:** Seamless integration with Azure services like Active Directory, monitoring, and security.
4. **Cost Efficiency:** Pay-as-you-go pricing for infrastructure, reducing operational costs.
5. **Security:** Built-in security features like Azure Active Directory integration and network policies ensure robust protection.

AKS simplifies container orchestration, enabling organizations to deploy highly scalable, reliable, and secure applications on the cloud.