Journey to DevOps Nirvana: Automation & AI Integration

Submitted by:
SAMBHAJI IPPAR
885865899

California State University, Fullerton
CPSC-597
Fall 2023
Advisor: Dr. Sampson Akwafuo

Department of Computer Science
California State University Fullerton

Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form

This project report is acceptable in partial fulfilment of the requirements for the award of Master of Science degree in Computer Science.

Journey to DevOps Nirvana: Automation & AI Integration

Project Title

Sambhaji Shashikant Ippar

Student Name

Dr. Sampson Akwafuo

Advisor's Name

Advisor's signature Date

Reviewer's name

Reviewer's signature Date

Table of Contents

Table of Contents

Table of Figures

Abstract

DevOps has gained prominence for its emphasis on teamwork, automation, and integration between development and IT operations teams. However, the proliferation of DevOps tools and diverse cloud providers has introduced challenges in migration and skills transfer. As software systems become more complex, DevOps can encounter issues such as high costs, error prediction difficulties, complex integrations, and prolonged build and release times.

Agile practices have become popular due to their ability to provide flexibility, efficiency, and speed in the Software Development Life Cycle (SDLC). Implementing a CI/CD pipeline within an Agile framework accelerates software delivery and boosts productivity[1].

Continuous Integration (CI) and Continuous Delivery (CD) practices, introduced by Martin Fowler and extended by J. Humble and D. Farley, play a pivotal role in reducing risks, ensuring software quality, and enabling frequent Delivery. The benefits of CI/CD include accelerated time-to-market, improved product quality, enhanced customer satisfaction, reliable releases, and increased productivity.

As most software development leverages Infrastructure-as-a-Service (IaaS), CI/CD has become essential in cloud computing. Nevertheless, even well-implemented systems may revert to previous versions due to post-deployment performance issues. This work addresses key challenges related to benchmarking systems with minimal impact, conducting load tests for reliable results, and streamlining these tests[1].

The transition from traditional software development methodologies to Agile practices and CI/CD pipelines is driven by the need for faster, more efficient software delivery. The role of automation, tools, and technologies in this transformation is paramount. Automation mitigates human errors, and tools like Terraform automate the Deployment of cloud resources. Moreover, secure storage of state files and sensitive data through tools like HashiCorp Cloud and Vault enhances the overall security of the CI/CD pipeline[2].

The paper further discusses the challenges in the current CI/CD pipeline, including manual processes, complex integrations, lengthy testing cycles, bottlenecks, resource provisioning, and error handling. These challenges contribute to a slower deployment cycle, impacting time-to-market, risk, productivity, and costs.

In conclusion, the paper highlights the motivations for this project, emphasizing the importance of overcoming human errors, and security vulnerabilities, and addressing current challenges. The

proposed project aims to enhance the CI/CD pipeline by leveraging artificial intelligence to detect and rectify errors, bolstering security measures, and streamlining the deployment process. This project is poised to contribute significantly to the ongoing evolution of software development processes and the improvement of CI/CD pipelines[3]

1. Introduction

Software development is constantly changing, so it's vital to continually improve processes for fast, reliable, and affordable software deployments. DevOps is a way of working that's become popular because it promotes teamwork, automation, and integration between development and IT operations teams. We still need to establish a standard for CI/CD[3]. This broader perspective on DevOps means hundreds of DevOps tools are available, making it difficult to migrate from one technology to another. Additionally, it's challenging to transfer skills learned in one tool to another. Furthermore, there are various cloud providers, with AWS, Google Cloud, and Azure being the top three. Learning the cloud is another daunting task in this development process.

This approach helps improve software development quality and reduce manual interventions. However, as software systems get more complicated, DevOps can sometimes run into problems like high costs, difficulty predicting errors, complex integrations, and long build and release times.

Nowadays, traditional software development methodologies are no longer enough to meet modern business requirements. Agile practices have become popular among software development companies as they enable flexibility, efficiency, and speed in the Software Development Life Cycle (SDLC). The Agile Manifesto has defined twelve principles that ensure the integrity of processes and practices in Agile Project Management[4]. These principles can be applied to methodologies such as Extreme Programming (XP), Scrum, Kanban, Crystal, Lean Software Development (LSD), and Feature-Driven Development (FDD)[5]. By implementing a CICD pipeline on Agile, companies can deliver software faster and increase productivity.

In 2000, Martin Fowler presented the idea of Continuous Integration (CI), which J.Humble and D.Farley later extended into the approach of Continuous Delivery (CD) as a concept of the deployment pipeline. The main benefit of CI practices is reducing the risk and making software bug-free and reliable, which removes the barrier to frequent Delivery. Accelerated time to market, improved product quality, improved customer satisfaction, reliable release, improved productivity, and efficiency are key benefits that motivate companies to invest in CD.

Since most software and mobile developments are deployed on infrastructure-as-a-service (IaaS), CICD has become an essential part of cloud computing. However, even if a system is well-implemented and has passed holistic test cases, it may still revert to a previous version due to a performance issue identified after deployment[6]. The agile process places more concern on delivering the product within committed deadlines. The decision of system scale is an immediate solution for the issue.

The current system benchmark level and the new software benchmark level are two factors that define the target system scale size. System benchmarking can cause the system to be unstable since it has to be evaluated on production. Moreover, traditional load test simulations are highly deviated from production live traffic patterns, which produces unreliable results. Therefore, this work addresses the key problems of how to benchmark the system with minimum impact, how to perform load tests for more reliable results, and how to perform these tests with less effort.

When we are talking about Continuous integration and Continuous delivery/deployment, this are the following steps which come under that:

a. **Code Writing:** This is the first phase, where developers create new features, fix bugs, or improve the software. They can work alone or collaborate on larger projects using version control systems like Git.

b. **Code Commit:** Developers commit their code changes to a version control system like Git. This involves creating a snapshot of the code changes made during a specific task or feature development. Each commit is associated with a message that describes the purpose of the change. Commits are tracked and organized in the version control repository[7].

c. **Building Artifacts:** In this step, the committed code is compiled or built into deployable units such as executable artifacts, libraries, and more. The build process converts the source code into a format that can be deployed to a server or runtime environment. This can include compiling code, packaging files, and generating executable binaries.

d. **Testing Artifacts:** Once the artifacts are built, they are thoroughly tested to ensure that they work as expected. Testing can include unit tests, integration tests, and end-to-end tests. Automated testing helps catch bugs and regressions early in the deployment pipeline, reducing the chances of issues in production.

e. **Deploying Artifacts:** After successful testing, the built artifacts are deployed to a staging environment or a pre-production environment. This environment closely resembles the production environment and allows for final testing before the changes are deployed to the live production environment.

f. **Verifying Artifacts:** In the staging environment, artifacts are tested thoroughly to ensure they function correctly and do not introduce issues. This includes additional testing specific to the staging environment to simulate real-world usage scenarios.

g. **Rolling Back Artifacts:** Sometimes, despite rigorous testing, issues may be discovered in the staging environment. In such cases, it's crucial to have a plan for rolling back to the previous stable software version. This can involve reverting to the prior set of artifacts, configuration, or data.

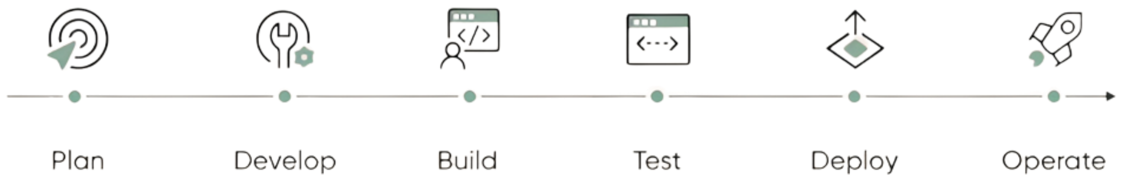


Figure 1 CICD (Continuous Integration/ Continuous Deployment) Phases

CICD Phases can be explained in detail as follows:

A. Continuous integration:

Continuous Integration (CI) is a fundamental aspect of modern software development practices. It replaces the traditional, manual, and error-prone method of code integration with an automated and systematic approach. The CI phase promotes a culture of continuous collaboration and integration among development teams, breaking down silos that can impede progress[7].

The principle behind CI is that code changes should be integrated into the main codebase frequently and early. This practice encourages developers to regularly push their code modifications to a shared repository, promoting transparency and making the latest changes easily accessible to all team members[7], [8].
The CI phase relies on automation to perform several crucial tasks. It starts with the automated build process, where code is compiled, dependencies are resolved, and a working version of the software is generated. Automated testing, both at the unit and integration level, is performed to ensure that the code remains functional and compatible as it evolves.

Code analysis tools are employed to maintain code quality, consistency, and security. This is particularly important in large and complex software projects, where maintaining standards and identifying potential issues can be challenging without automation. CI generates deployable artifacts, which are considered for Deployment in later stages. These artifacts are the outcome of thoroughly tested and validated code changes, not just a static snapshot of the code.

Reports generated throughout the CI process provide insights into the codebase's health, enabling teams to make informed decisions about the code's readiness for Deployment. The CI phase is equipped with an integrated notification system that alerts the development team in real time in the event of any issues. This immediate feedback mechanism helps teams respond quickly to problems, thus reducing the chances of integration conflicts and ensuring a consistent codebase[9].

B. Continuous Delivery:

Continuous Delivery (CD) is a crucial phase in the software development and delivery process. It is built on the foundation established by Continuous Integration (CI) and extends the principles of automation, collaboration, and quality assurance further downstream. The CD includes a set of practices and tools that aim to ensure that the software is always in a deployable state and ready for release to production or staging environments[8].

The primary objective of CD is to minimize the gap between code integration (in the CI phase) and reliable Deployment to end-users or other target environments. This entails automating various aspects of the software delivery pipeline, such as testing, Deployment, and infrastructure provisioning[7]. The CD phase ensures that every code change, no matter how small, goes through a standardized and streamlined process.

By automating the deployment process, CD not only accelerates the Delivery of new features or bug fixes but also reduces the risk of human error. It provides teams with the confidence that any code changes that have successfully passed through the CI phase are fit for Deployment. This confidence is derived from extensive testing, quality checks, and standardized deployment procedures.

During the CD phase, a new concept is introduced, called a "release candidate". This is a version of the software that has gone through multiple tests and checks, making it a strong contender for deployment. These release candidates undergo various testing levels, such as integration, performance, and user acceptance testing, to ensure they meet the required quality standards.

Integrating Continuous Delivery into the DevOps pipeline creates a reliable, repeatable, and efficient process for releasing software. It enables teams to respond to market demands, customer feedback, and business needs in a swift and controlled manner. The automation and consistency brought by CD contribute to reducing the overhead of manual tasks and decreasing the likelihood of deployment failures or rollbacks[10].

C. Continuous deployment cycle:

The Continuous Deployment (CD) phase is the most advanced stage in the DevOps pipeline. It represents a significant step towards near-instantaneous software delivery. CD builds upon the foundational principles of Continuous Integration (CI) and the systematic release management of Continuous Delivery (CD).
The CD takes automation, efficiency, and customer-centricity to new heights. It enables the automatic and swift Deployment of software to production environments. Unlike Continuous Delivery, where software is kept in a deployable state and release decisions are usually made manually, Continuous Deployment eliminates this manual intervention[11]. Every code change that successfully passes through the CI phase is automatically propagated through a series of rigorous tests and quality checks. Upon satisfying these criteria, the software is deployed to production without human intervention, creating an almost continuous stream of new releases.

The core philosophy of CD is rooted in the idea that software development should be a response to customer needs and market demands rather than a predetermined release schedule. It allows organizations to release new features, enhancements, or bug fixes as soon as they are ready, fostering an agile and highly responsive development process.

CD leverages automation for the deployment process, ensuring that deployments are consistent, repeatable, and error-free. This automation doesn't just apply to the application code but also extends to infrastructure provisioning, database updates, and other aspects of the deployment pipeline. CD embraces the idea of "infrastructure as code," where the entire software ecosystem is scripted and versioned, leading to more predictable and reliable deployments[12].

Continuous Deployment doesn't mean reckless or uncontrolled software releases. Rather, it emphasizes the importance of rigorous automated testing, monitoring, and validation. Automated tests are essential to catch any regressions or issues before they reach production. Real-time monitoring systems are also employed to detect and respond to potential issues in the production environment promptly.

Continuous Deployment fosters a culture of continuous learning and improvement. The immediate feedback from production deployments allows teams to assess the impact of their changes quickly. This feedback loop informs future development efforts, driving the software's evolution based on real-world usage and user feedback.

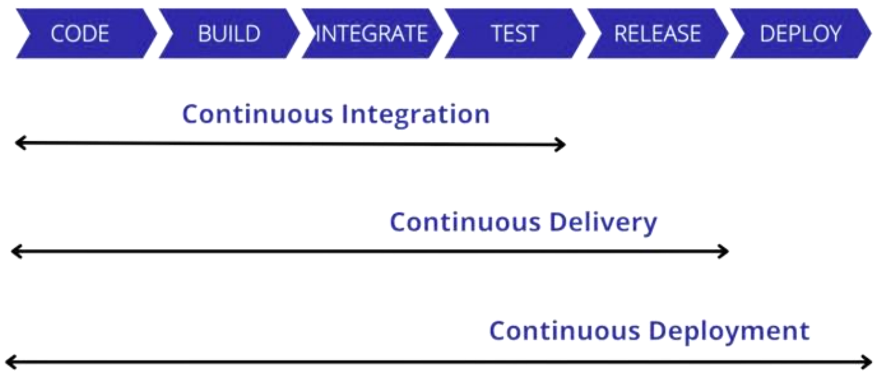


Figure 2 CICD Pipeline Stages

1.1 Motivation:

In today's world, where everything must be fast-paced, the need for continuous integration and continuous Delivery, which is fulfilled in every aspect, is important. Having a system that can withstand any type of security threats any type of attack and not prone to any vulnerability and uses best practices from the production to release software is essential. When it comes to providing speed into CICD pipeline main thing to avoid is human errors as those are hard to detect if not a proper analysis is done.

A. Human Errors:

Automation plays a crucial role in preventing human errors in the system. Infrastructure as a tool is supported by tools and technologies like Terraform. Terraform is important for the automation phase as it allows us to deploy cloud providers automatically. It supports hundreds of cloud providers and has numerous features, making it an ideal fit for the infrastructure as code tool for this project. Terraform also offers several features like HashiCorp cloud, which is a recommended solution for storing state files for security purposes. HashiCorp vault is another product by HashiCorp that comes in handy for encrypting sensitive information such as usernames, passwords, and secret keys and passing that data as variables when running the Terraform code.

B. Slower Deployment cycle:

Introduction to Current CI/CD Pipeline:

Begin by providing an overview of your existing CI/CD pipeline, emphasizing its purpose and significance in software development. Explain that the CI/CD pipeline is designed to automate the process of building, testing, and deploying software, but it's facing challenges in terms of deployment speed[13].

C. Factors Contributing to Slower Deployment Cycle:

a. Manual Processes: One of the primary reasons for the slower deployment cycle is the presence of manual processes within the pipeline. Manual interventions, such as manual code reviews or manual approvals, can introduce delays as they rely on human availability and decision-making. This manual intervention may be necessary for quality control but can slow down the process.

b. Complex Integration: In a complex software environment, integrating various components, services, and third-party tools can be challenging. Integrations between different stages of the pipeline may not be seamless, leading to delays in the deployment process. The need for extensive testing and validation of these integrations can also slow things down[14].

c. Long Testing Cycles: Thorough testing is essential for ensuring software quality. However, lengthy testing cycles can significantly slow down the deployment process. Testing might involve unit tests, integration tests, and end-to-end tests, and the time taken for these tests can accumulate.

d. Bottlenecks: Identify specific bottlenecks within the pipeline. These can be resource limitations, hardware constraints, or inefficient resource allocation. Bottlenecks can cause certain stages of the pipeline to wait for resources, leading to a slower overall deployment cycle.

e. Resource Provisioning: The process of provisioning infrastructure resources for Deployment, especially in cloud environments, can be time-consuming. If your pipeline relies on manual provisioning of resources, it can lead to delays.

f. Error Handling: The pipeline might not be equipped to efficiently handle errors and exceptions. When errors occur during the deployment process, it can lead to rollbacks, debugging, and retesting, all of which extend the deployment time.

D. Impact of Slower Deployment Cycle:

Consequences of a slower deployment cycle, such as:
Increased time-to-market for new features and updates, which can negatively impact competitiveness.
DevOps deployment cycles that are slow can be responsible for several significant consequences, which can affect both the organization as a whole and the development process. These consequences encompass:

a. **Reduced Agility:** The ability of an organization to respond quickly to changing market demands or customer feedback is hindered by slower deployment cycles. It limits the agility of the development team and the business.
b. **Delayed Time-to-Market:** Longer deployment cycles result in more time being taken to introduce new features, products, or updates to customers. This can result in missed business opportunities and a competitive disadvantage.
c. **Increased Risk**: Introducing errors, security vulnerabilities, and compliance issues poses a greater risk the longer the time between code development and Deployment. Delayed deployment cycles can result in more extensive and riskier code releases.
d. **Reduced Productivity:** Developers spend more time waiting for code to be deployed than writing new code or addressing critical issues. This can lead to reduced developer productivity and job dissatisfaction.
e. **Higher Cost of Development:** Inefficiencies often result in lengthy deployment cycles, increasing the development cost. The longer it takes to deliver a product, the more resources are consumed, impacting both time and financial resources[15].
f. **Customer Frustration:** Slow deployment cycles can lead to customer frustration, churn, and negative feedback, damaging the organization's reputation. Customers expect timely updates and improvements.
g. **Quality Issues:** Inadequate testing and quality assurance may result from rushing to meet deadlines in a slow deployment cycle. This can lead to more post-deployment issues and customer complaints.
h. **Complex Merges and Conflicts:** Accumulating more code changes before Deployment due to longer cycles can result in complex and time-consuming merges, conflicts, and code integration issues.
i. **Inefficient Resource Allocation:** Waiting for deployments can result in team idling, which is an inefficient allocation of resources. This can lead to unproductive downtime.
j. **Difficulty in Scaling:** Maintaining a slow deployment cycle can become unsustainable as an organization grows. Processes that need to be more streamlined pose challenges in scaling development and operations.
k. **Impaired Collaboration:** Slow deployment cycles can strain collaboration between development and operations teams. Efficient coordination and communication become more challenging.
l. **Compliance Challenges:** Keeping track of changes and ensuring they meet compliance standards becomes more complex, posing difficulties in maintaining compliance with regulatory requirements due to slower deployments.

To put it simply, slower DevOps deployment cycles can have significant consequences beyond development. They can affect an organization's agility, competitiveness, and ability to adapt to changes in market conditions and customer expectations. Therefore, it is highly beneficial for organizations to streamline their deployment cycles and embrace faster, more efficient DevOps practices.[16]

These are the main factors that motivated the project to come into reality. In the last 20 years there has been a lot of improvement in the CICD (Continuous integration Continuous delivery) process. However, there is still space for more improvements.

The current development and operational team must go through the following problems:

1. **Delay in finding and fixing errors:**
   When we test new features, they will be prone to different errors, which take longer to detect and fix. Whenever there is an error, we must go through the analysis phases and highlight the actual reason for the error, which takes longer than expected time from the development as it slows the building process. Even if it's a simple misspelling in the code, it will take a lot of time out of the developers. Which on a large scale hurts the production and creates a delay. To mitigate this, we can take advantage of Artificial Intelligence.

2. **Security**
   Security is the main factor in DevOps as every tool uses different ports and security policies, so handling security along with the latest patches and bug fixes is important.
   Common Security Problems in the Current Market:

   a. **Insecure Dependencies:** Many security breaches result from vulnerabilities in third-party dependencies. Ensuring that dependencies are regularly updated and free of known vulnerabilities is essential.
   b. **Insufficient Access Control:** Failing to implement proper access controls can lead to unauthorized access and data breaches. Implement RBAC and least privilege principles[14].
   c. **Lack of Code Analysis:** Without proper code analysis, security vulnerabilities can go unnoticed until the deployment phase, increasing the cost and effort of remediation.
   d. **Inadequate Testing:** Insufficient or poorly executed security testing can result in undetected vulnerabilities. Comprehensive and automated security testing is critical.
   e. **Ignoring Secrets Management:** Inadequate management of secrets and sensitive data can lead to data breaches and unauthorized access to critical systems[17].
   f. **Configuration Errors:** Misconfigurations in application and infrastructure settings can expose security weaknesses. Continuous monitoring and auditing are vital to identifying and rectifying such issues.
   g. **Outdated Software:** Failing to keep software, libraries, and dependencies up to date can result in security vulnerabilities that attackers can exploit.

1.2 Objectives

Project Objectives:

1. **Operational Cost Optimization:** In the currently running system, there is no way for us to optimize the cost of Kubernetes deployment internally, as this is a cumbersome process and requires experienced developers to run a best practice and run a cost optimization on ec2 instances based on their experience[18]. However, imagine developers repeating the same thing multiple times a week. Because there will always be new deployments running and monitoring the deployments based on usage, optimizing those deployments to save cost takes a lot of effort and creates an overhead on the DevOps team[19] with Implementation of AI-driven automation to reduce operational costs associated with resource allocation, infrastructure provisioning, and cloud services. Operational cost optimization can be achieved.

Whenever we are talking about the production level, there are hundreds of or maybe thousands of servers running and testing different features in different teams. In that scenario, if we are using any cloud providers to provision those resources, in our case EC2, then the cost of running 500 EC2 instances for a month will be:

The cost of running 500 Amazon EC2 instances for a month can vary depending on several factors, including the instance type, region, and any additional services or features you might use. In our case, we are going to need at least t2. Medium EC2 servers.

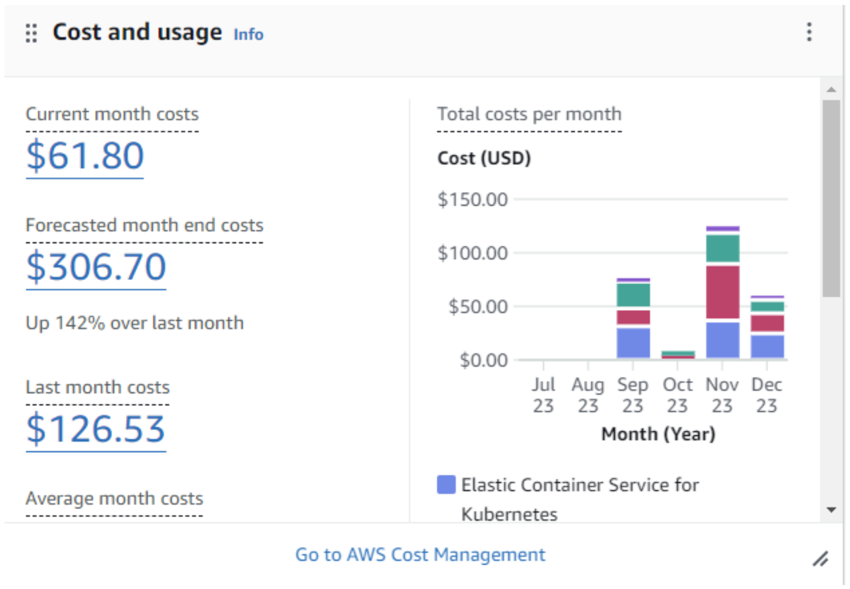


Figure 3 Current resource cost on the AWS

Specification of t2.medium EC2:

MEMORY(GiB): 2
vCPU: 4
Storage: 20GB (Instance store)

Instance store:
An instance store is a type of temporary block-level storage that is physically attached to the host machine of an Amazon EC2 instance. This storage is non-persistent, which means that any data stored on the disk is lost when the instance is either stopped or terminated. Ideal for temporary data, scratch space, and other non-critical data, instance store volumes are not required to persist beyond the life of the instance.[20]

As of my last update in September 2021, the on-demand pricing for a t2.medium instance in the US East (N. Virginia) region was approximately $0.0116 per hour.
Let's do some calculations:

- Cost per t2.medium instance per hour: $0.26
- Number of instances: 500
- Hours in a month: 24 * 30 (assuming 30 days in a month)

Now, calculate the total cost:
Total Cost = (Cost per instance per hour) * (Number of instances) * (Hours in a month)
Total Cost = $0.268* 500 * (24 * 30)
Total cost ≈ $11,570.50$

This is just a rough estimate for the EC2 instance. Even if we can reduce that cost to 20% with the AI optimization, the cost will be closer to 9256$, which will save a total of 2314$.

- **Error Prediction and Prevention:** Develop AI models for proactive error prediction, preventing production outages, and ensuring the overall reliability of the CI/CD pipeline.
- **Seamless Tool Integration:** Successfully integrate multiple DevOps tools, including Terraform, GitHub, Jenkins, Docker, Kubernetes, SonarQube, JFrog Artifactory, Helm, Prometheus, and Grafana to establish a coherent CI/CD pipeline.
- **Efficiency and Speed:** Improve software build and release times by seamlessly integrating DevOps tools and incorporating AI-driven automation for faster pipeline deployment.
- **AI-Enabled Cost Reduction:** Apply AI techniques to optimize cloud infrastructure costs, identifying opportunities to reduce expenses while maintaining performance.
- **Detailed Monitoring and Reporting:** Establish comprehensive monitoring and reporting capabilities using tools such as Prometheus and Grafana to gain insights into system performance, resource usage, and error trends.
- **Scalable Infrastructure:** Architect infrastructure with Terraform and Kubernetes to enable easy scalability, accommodating varying workloads efficiently.
- **Reusable Infrastructure as Code (IaC):** Create Infrastructure as Code (IaC) scripts with a focus on reusability, enabling swift provisioning of infrastructure for multiple projects and environments[21].
- **High Availability and Redundancy:** Configure Kubernetes orchestration to prioritize high availability and redundancy, minimizing downtime and ensuring dependable service delivery.
- **Enhanced Deployment Speed:** Streamline the CI/CD pipeline for rapid and efficient Deployment of software updates and features to production environments.

1.3 Contributions

The project makes a substantial contribution to the field of DevOps and Artificial Intelligence (AI) integration, addressing critical challenges and pushing the boundaries of existing practices. This project seeks to advance the current state of DevOps by seamlessly integrating AI technologies into the pipeline, thus offering several significant contributions:

1. Enhanced DevOps Efficiency through AI Integration: The project significantly contributes to the domain by exploring the potential of AI integration into DevOps. By leveraging machine learning and predictive analytics, it aims to minimize operational costs and anticipate potential errors. This integration is essential for overcoming the challenges associated with traditional DevOps practices, such as high costs and difficulties in predicting and preventing errors.[22]

2. Cutting-Edge Infrastructure and Tool Integration: In the realm of DevOps, the project contributes by incorporating and integrating a range of industry-standard tools and technologies, such as Terraform, Jenkins, Docker, Kubernetes, SonarQube, Prometheus, and Grafana, among others. The comprehensive analysis and justification for the selection of these tools are essential for guiding developers and organizations in choosing the most suitable tools for their DevOps practices.

3. AI-Driven Error Prediction: The project focuses on active error prediction using AI, a groundbreaking approach. This contribution has the potential to revolutionize DevOps practices by reducing production outages and enhancing system reliability. By identifying potential issues before they escalate, the project aims to minimize the occurrence of production incidents and reduce downtime.

4. Detailed Monitoring and Reporting: The incorporation of detailed monitoring and reporting capabilities through tools like Prometheus and Grafana contributes to the effective management of DevOps processes. It provides insights into system performance, resource usage, and error trends, thereby improving the decision-making process for software delivery[23].

5. Scalable Infrastructure and Reusable IaC: The project's focus on scalable infrastructure and reusable Infrastructure as Code (IaC) scripts, developed using Terraform and Ansible, makes a valuable contribution. This approach allows for swift and effective provisioning of infrastructure for multiple projects and environments, improving resource allocation and utilization.

6. High Availability and Redundancy: Prioritizing high availability and redundancy in the configuration of Kubernetes orchestration is essential for minimizing downtime and ensuring dependable service delivery. This contribution adds to the resilience and reliability of the DevOps pipeline.

This project makes a significant contribution to the field of DevOps by addressing complex challenges, enhancing operational efficiency, and pushing the boundaries of traditional practices. By seamlessly integrating AI and automation, this project aims to improve software development quality, reduce operational costs, and anticipate and prevent potential errors, thus revolutionizing DevOps practices for the future[24].

2. Literature Review

Traditional software development methodologies posed various challenges that led to the emergence of DevOps. The Waterfall model, characterized by its sequential approach, hindered the adaptability required to meet rapidly evolving business requirements. Agile practices introduced flexibility but often lacked streamlined deployment processes, leading to bottlenecks in software delivery. DevOps principles seek to enhance software development processes with its key principles outlined in the Agile manifesto. It emphasizes collaboration, automation, continuous integration, continuous Delivery, and rapid feedback cycles.

2.1 Automation and Security in DevOps:

The use of automation is crucial in resolving the issues related to DevOps. Tools such as Terraform offer solutions for automating the Deployment of cloud resources, helping to eliminate manual errors and expedite the deployment process. Terraforms extensive features, including the secure storage of state files and sensitive data using tools like HashiCorp Cloud and Vault, contribute to the overall security of the DevOps pipeline.

Security remains a central concern in DevOps. Insecure dependencies, insufficient access control, code analysis, and inadequate testing can lead to vulnerabilities. The importance of secrets management, configuration error prevention, and staying updated with software patches and dependencies cannot be understated. Security practices in DevOps continue to evolve in response to the ever-increasing threat landscape[25].

Continuous Integration (CI) and Continuous Delivery (CD) are the cornerstones of DevOps. CI aims to minimize the risks associated with software development by automating the process of code integration and verification through the use of automated builds, unit tests, and integration tests. It seeks to ensure that code changes are frequently and reliably integrated into a shared repository, promoting transparency and reducing integration conflicts.

Continuous Delivery builds upon CI by automating the entire software delivery pipeline. It extends the principles of automation, collaboration, and quality assurance further downstream. CD aims to ensure that the software is always in a deployable state and ready for release to production or staging environments[26]. By automating various aspects of the software delivery pipeline, continuous Delivery (CD) accelerates the Delivery of new features and bug fixes while reducing the risk of human error.

2.2 Challenges and Motivation for Improvement

While DevOps has brought significant advancements to software development processes, it is not without its challenges. The complexity of modern software systems, a profusion of tools and technologies, and a multitude of cloud providers have introduced challenges in migration and skills transfer. Organizations often encounter high costs, difficulties in predicting errors, complex integrations, and prolonged build and release times.

The motivation to enhance DevOps practices stems from these challenges and a quest for continuous improvement. These challenges have prompted the need for solutions that streamline benchmarking systems, improve load testing for more reliable results, and minimize the impact of system benchmarking on production environments.

2.3 Automation and Security in DevOps

The use of automation is crucial in resolving the issues related to DevOps. Tools such as Terraform offer solutions for automating the Deployment of cloud resources, helping to eliminate manual errors and expedite the deployment process. Terraforms extensive features, including the secure storage of state files and sensitive data using tools like HashiCorp Cloud and Vault, contribute to the overall security of the DevOps pipeline.

Security remains a central concern in DevOps. Insecure dependencies, insufficient access control, code analysis, and inadequate testing can lead to vulnerabilities[27]. The importance of secrets management, configuration error prevention, and staying updated with software patches and dependencies cannot be understated. Security practices in DevOps continue to evolve in response to the ever-increasing threat landscape.

DevOps is a transformative software development approach driven by collaboration, automation, and efficiency. Continuous Integration and Continuous Delivery, as foundational practices, aim to reduce risks, enhance software quality, and expedite software delivery[28]. The challenges and motivation for improvement have led to the adoption of automation and security practices, offering solutions to streamline and secure the DevOps pipeline. As the DevOps landscape evolves, these principles and practices continue to shape and redefine software development processes.

3. Methodology

The integration of AI with DevOps is a compelling area of research and project focus, driven by several critical factors. The IT world is undergoing a significant transformation due to the increasing adoption of DevOps practices, which aim to unite development and operations teams, resulting in faster and more efficient software development and deployment. However, one of the foremost challenges in DevOps is identifying and mitigating errors in real-time to ensure the continuous delivery of high-quality software. Additionally, optimizing operational costs in DevOps environments is vital for sustainable business operations. This research aims to comprehensively address these challenges by combining AI with DevOps.

There is a noticeable gap in the literature regarding the practical implementation of AI solutions in DevOps for error prediction and cost optimization. While existing studies often provide theoretical frameworks, applicable guidelines and insights are limited. This project aims to bridge this gap and offer a methodical approach to effectively implementing AI in DevOps.

Integrating AI in DevOps involves using machine learning (ML) and various artificial intelligence technologies to enhance and automate software development and delivery tasks. This integration aids in software testing, preparation, security, and resource management. AI in DevOps allows organizations to experience notable improvements in the software development lifecycle's efficiency, accuracy, and reliability, leading to swifter deployments, reduced errors, and increased overall productivity[29]. AI can help manage cloud resources for DevOps by predicting future needs and adjusting resources to optimize cost and performance. AI-powered cloud management platforms can analyze resource utilization, performance, and cost across multiple cloud providers, enabling organizations to maximize resource allocation, streamline deployments, enhance security, and improve performance monitoring. Embracing AI for infrastructure as code (IaC) practices empowers teams to automate and intelligently manage infrastructure, resulting in faster time-to-market, cost savings, and increased productivity.

Optimizing cloud costs for DevOps with AI-assisted orchestration can lead to significant cost savings. An AI-assisted Kubernetes orchestrator can eliminate up to 60% of cloud costs by being smarter about the use of cloud resources and augmenting Kubernetes with AI tooling. This smarter, more responsive approach to optimizing cloud compute resource orchestration is essential for DevOps, DevSecOps, and SRE use cases, enabling organizations to achieve optimizations at the speed needed for their operations.[30]

The incorporation of AI in DevOps can result in enhanced speed, improved resource management, increased reliability throughout the software development lifecycle, and more efficient operations. AI assists DevOps teams in refining their operations by detecting inefficiencies, triggering warnings as soon as issues appear, and enabling real-time observation of systems and applications, minimizing downtime. [31]AI can be utilized to make DevOps processes more efficient by creating test cases and pipelines, assessing infrastructure configurations for security concerns, and deploying applications.

In conclusion, integrating AI with DevOps for error prediction and cost optimization is a promising area that can significantly enhance software development and deployment's efficiency, reliability, and cost-effectiveness. By leveraging AI technologies, organizations can automate and intelligently manage their operations, leading to faster time-to-market, reduced errors, and increased overall productivity. [32]However, addressing the challenges and limitations of AI integration in DevOps is essential to realize its full potential.

3.1 Toolset and Implementation

Terraform:

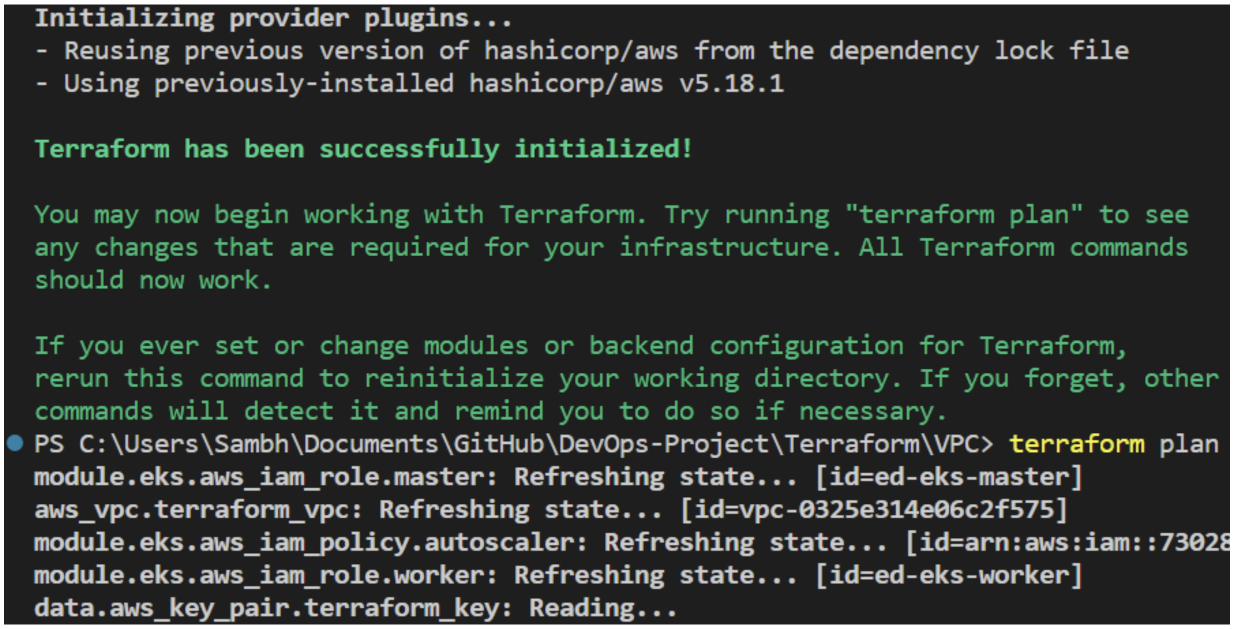
Terraform is an infrastructure as a code tool that automates the process of defining, provisioning, and managing infrastructure resources. It operates by allowing users to describe the desired state of their infrastructure in configuration files. Terraform supports on-premises infrastructure and major cloud providers.

Key Features of Terraform:

- **Declarative Syntax:** Terraform uses a declarative syntax to describe the infrastructure state, making it easy to read and understand.
- **Infrastructure as Code:** Terraform configuration files are written in a domain-specific language (HCL) or JSON, which enables version control, testing, and collaboration on infrastructure code.
- **State Management:** Terraform keeps track of the current state of the infrastructure, allowing it to plan and apply only the necessary changes, reducing the risk of unintended modifications.
- **Resource Providers:** Terraform supports a wide range of providers, including cloud services, databases, and networking components, enabling resource management across various cloud providers and on-premises.
- **Modularity:** Terraform simplifies infrastructure management by breaking configs into modular components.
- **Immutable Infrastructure:** Terraform encourages the creation of immutable infrastructure, reducing configuration drift and ensuring consistent environments.

Benefits of Terraform in DevOps

- **Automation:** Terraform automates the provisioning and management of infrastructure, reducing manual tasks, minimizing errors, and enabling continuous integration and continuous delivery (CI/CD) pipelines.
- **Scalability:** Terraform enables you to easily scale your infrastructure up or down to meet changing demands, handling infrastructure growth efficiently.
- **Version Control:** Infrastructure as code can be versioned, allowing teams to track changes, collaborate effectively, and roll back to previous states if necessary.
- **Portability:** Terraforms multi-cloud support enables infrastructure creation that is not tied to a specific cloud provider, providing flexibility and reducing vendor lock-in.
- **Cost Management:** DevOps teams can use Terraform to optimize infrastructure costs by automating resource provisioning and de-provisioning, ensuring that only necessary resources are running.
- **Security and Compliance:** By defining infrastructure in code, it's easier to apply security policies and ensure compliance by design, reducing security risks[31].
- **Rapid Deployment:** Terraform enables rapid infrastructure deployment, reducing lead times and accelerating development cycles.



```
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.18.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\Sambh\Documents\GitHub\DevOps-Project\Terraform\VPC> terraform plan
module.eks.aws_iam_role.master: Refreshing state... [id=ed-eks-master]
aws_vpc.terraform_vpc: Refreshing state... [id=vpc-0325e314e06c2f575]
module.eks.aws_iam_policy.autoscaler: Refreshing state... [id=arn:aws:iam::73028
module.eks.aws_iam_role.worker: Refreshing state... [id=ed-eks-worker]
data.aws_key_pair.terraform_key: Reading...
```

Figure 4 Terraform code running

Use Cases:

Terraform is used in a wide range of use cases, such as:

- **Provisioning Cloud Resources:** Creating and managing virtual machines, databases, storage, and networking resources on cloud providers.

- **Hybrid Cloud Deployments:** Managing infrastructure across both on-premises data centers and cloud environments.
- **Application Environments:** Defining and deploying infrastructure for applications, including development, testing, staging, and production environments.
- **Multi-Tier Architectures:** Building complex, multi-tiered application architectures with load balancers, databases, and web servers.

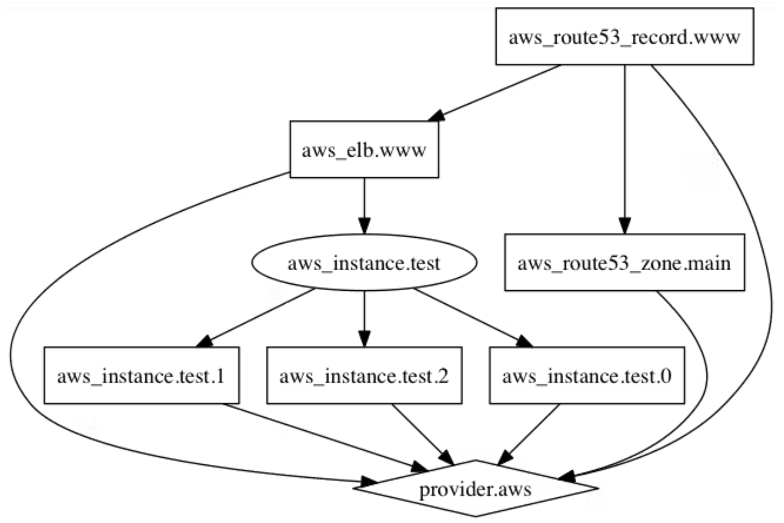
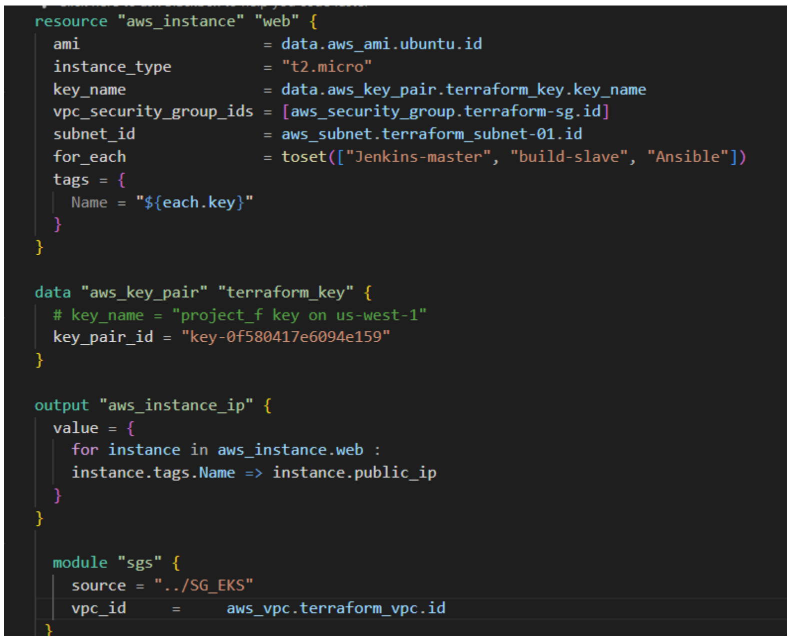


Figure 5 Terraform architecture implementation graph



```
resource "aws_instance" "web" {
  ami                    = data.aws_ami.ubuntu.id
  instance_type          = "t2.micro"
  key_name               = data.aws_key_pair.terraform_key.key_name
  vpc_security_group_ids = [aws_security_group.terraform-sg.id]
  subnet_id              = aws_subnet.terraform_subnet-01.id
  for_each               = toset(["Jenkins-master", "build-slave", "Ansible"])
  tags = {
    Name = "${each.key}"
  }
}

data "aws_key_pair" "terraform_key" {
  # key_name = "project_f key on us-west-1"
  key_pair_id = "key-0f580417e6094e159"
}

output "aws_instance_ip" {
  value = {
    for instance in aws_instance.web :
    instance.tags.Name => instance.public_ip
  }
}

  module "sgs" {
    source = "../SG_EKS"
    vpc_id     =     aws_vpc.terraform_vpc.id
  }
```

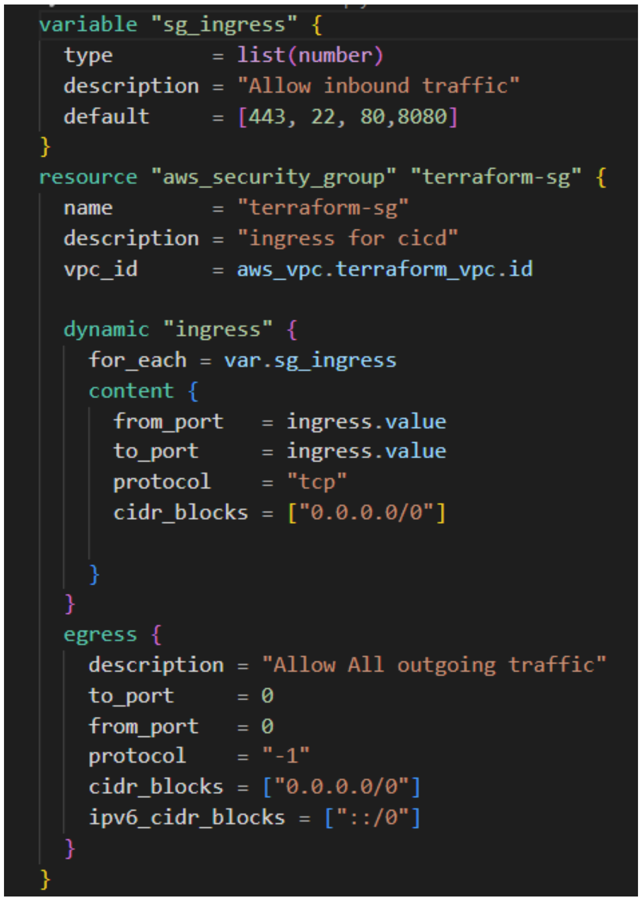
Figure 6 Terraform Code to provision AWS instances

The code snippet provided is written in Terraform and is primarily used to create instances that run on the Ubuntu operating system. These instances are provisioned to create different servers that are required for the project. The servers created include:

1. Jenkins server and Maven server, where Docker will also be installed. The Jenkins server is used for continuous integration and continuous deployment (CI/CD) of the project. The Maven server is used to manage dependencies and build the project. Docker is installed on both servers to enable containerization of the project.

2. Maven Slave, which is responsible for creating an artifact for our Java application to run with its dependencies. The Maven Slave is a lightweight instance that is used to offload the build process from the Jenkins server.

3. Ansible server, which is used to create an Ansible playbook to set up Jenkins. This includes installing Jenkins, Java, and turning on the services. All of this can be done with the help of the Terraform code mentioned above. The Ansible server is also used to manage the configuration of the Jenkins server and ensure that it is always up to date.

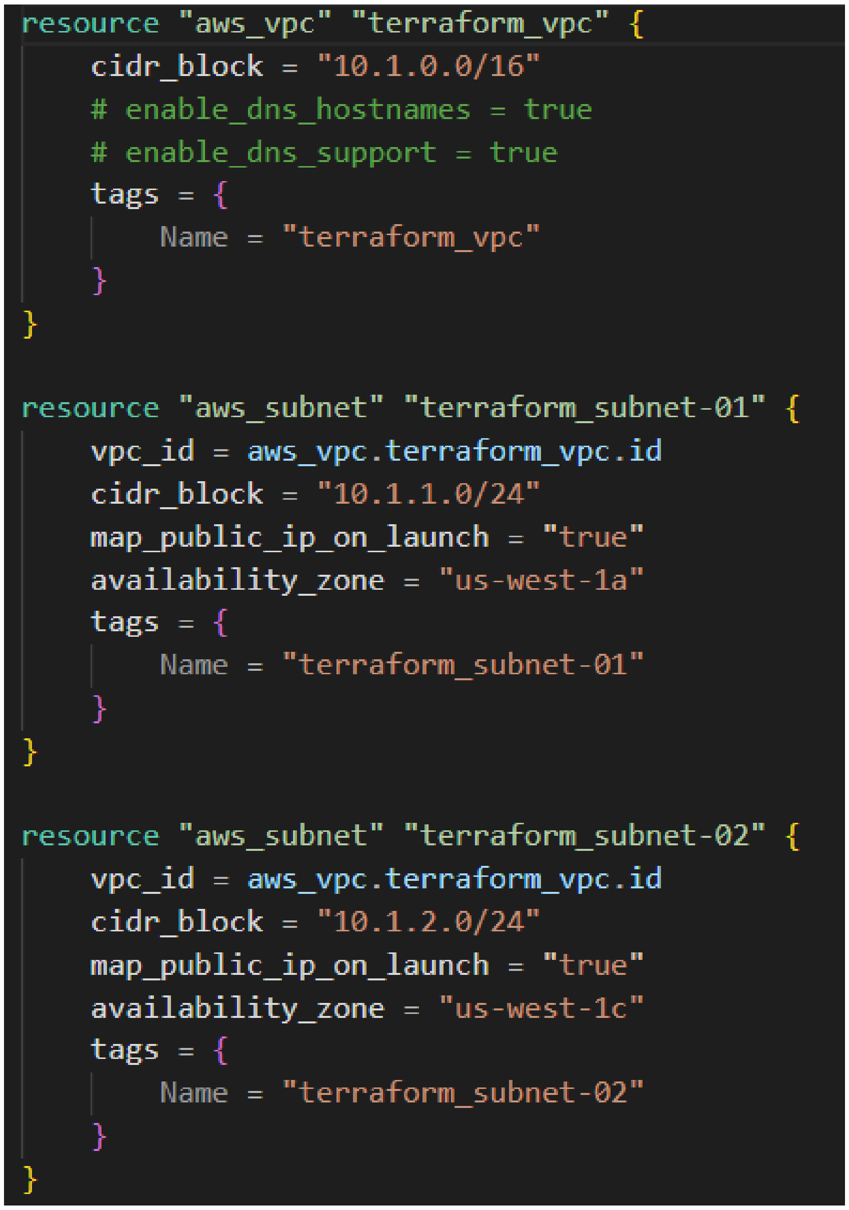


```
variable "sg_ingress" {
  type        = list(number)
  description = "Allow inbound traffic"
  default     = [443, 22, 80,8080]
}
resource "aws_security_group" "terraform-sg" {
  name        = "terraform-sg"
  description = "ingress for cicd"
  vpc_id      = aws_vpc.terraform_vpc.id

  dynamic "ingress" {
    for_each = var.sg_ingress
    content {
      from_port   = ingress.value
      to_port     = ingress.value
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]

    }
  }
  egress {
    description = "Allow All outgoing traffic"
    to_port     = 0
    from_port   = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}
```

Figure 7 Terraform Code to create Security Groups

We are using Terraform code to create a security group that will restrict traffic for our DevOps pipeline security. The dynamic block is created to take advantage of the dynamic function, which allows us to call a dictionary to allow multiple ports at the same time. These ports are used to access specific services on our cluster. We have allowed port number 22, which is used for SSH (Secure Shell) into our instances, along with port 443 for HTTPS traffic and port 8080 for web servers.

The egress rule is set to allow all outgoing traffic from the instances since there are no limitations on the outgoing traffic.

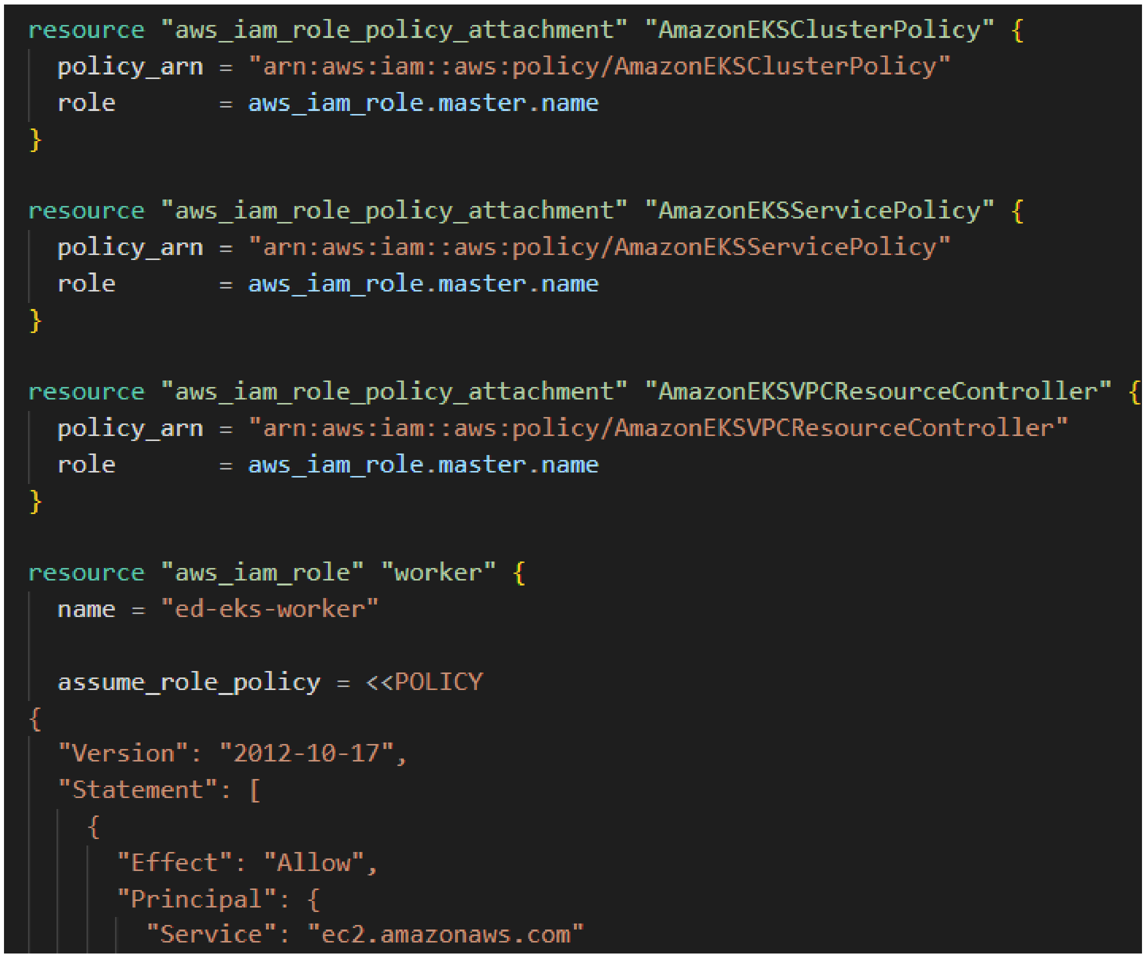


```
resource "aws_vpc" "terraform_vpc" {
    cidr_block = "10.1.0.0/16"
    # enable_dns_hostnames = true
    # enable_dns_support = true
    tags = {
        Name = "terraform_vpc"
    }
}

resource "aws_subnet" "terraform_subnet-01" {
    vpc_id = aws_vpc.terraform_vpc.id
    cidr_block = "10.1.1.0/24"
    map_public_ip_on_launch = "true"
    availability_zone = "us-west-1a"
    tags = {
        Name = "terraform_subnet-01"
    }
}

resource "aws_subnet" "terraform_subnet-02" {
    vpc_id = aws_vpc.terraform_vpc.id
    cidr_block = "10.1.2.0/24"
    map_public_ip_on_launch = "true"
    availability_zone = "us-west-1c"
    tags = {
        Name = "terraform_subnet-02"
    }
}
```

Figure 8 Terraform code to create VPC

In Figure 7, the code creates a Virtual Private Cloud (VPC) in our AWS provisioner. The purpose of creating a private cloud is to transfer data internally between services and protect against public cloud attacks, as well as man-in-the-middle attacks. [32]VPCs allow us to share intranet information within the network.

In addition to the VPC, two subnets are created for high availability and load balancing purposes. This ensures that even if one availability zone fails, our services will not be interrupted. The code uses the us-west-1 region to create availability zones.

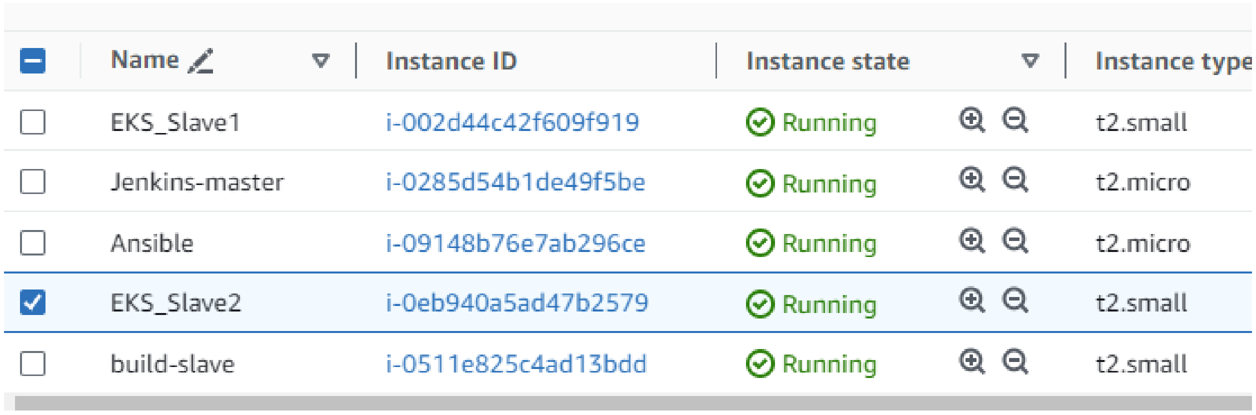


```
resource "aws_iam_role_policy_attachment" "AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.master.name
}

resource "aws_iam_role_policy_attachment" "AmazonEKSServicePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
  role       = aws_iam_role.master.name
}

resource "aws_iam_role_policy_attachment" "AmazonEKSVPCResourceController" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
  role       = aws_iam_role.master.name
}

resource "aws_iam_role" "worker" {
  name = "ed-eks-worker"

  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
```

Figure 9 Terraform Code for Creating Security Policies

In Figure 8, we attached EKS policies to allow smooth communication and access to EC2 instances.

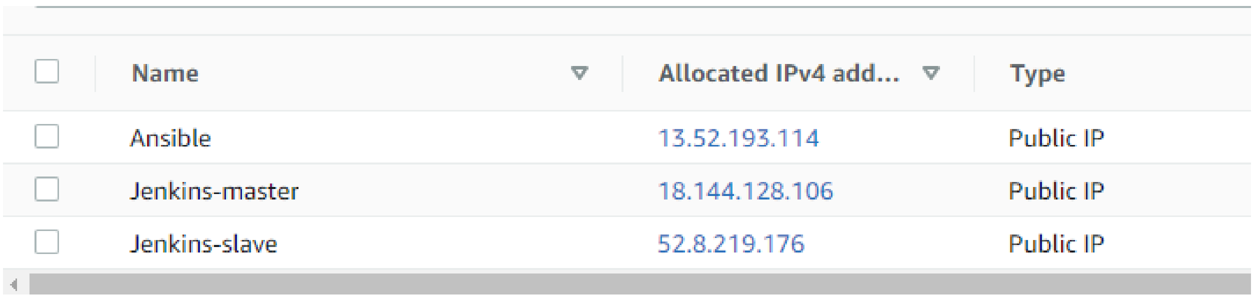
Amazon Web Services:



	Name	Instance ID	Instance state	Instance type
☐	EKS_Slave1	i-002d44c42f609f919	Running	t2.small
☐	Jenkins-master	i-0285d54b1de49f5be	Running	t2.micro
☐	Ansible	i-09148b76e7ab296ce	Running	t2.micro
✓	EKS_Slave2	i-0eb940a5ad47b2579	Running	t2.small
☐	build-slave	i-0511e825c4ad13bdd	Running	t2.small

Figure 10 AWS Dashboard after provisioning instances with Terraform

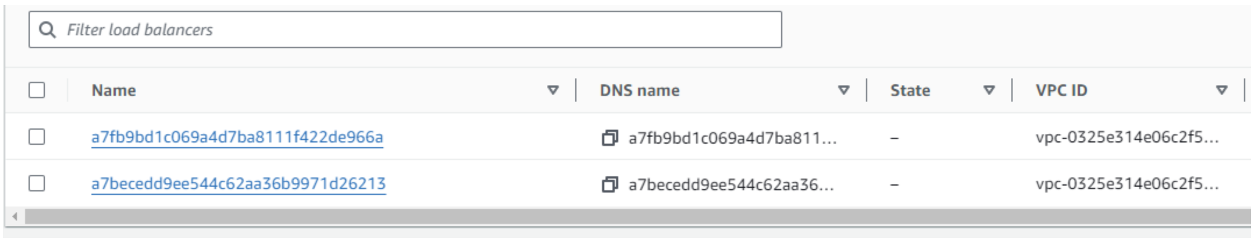
Figure 9 displays the instances that we are currently running on AWS. These instances were provisioned using Terraform code. EKS Slave1 and EKS Slave2 are servers that run Elastic Kubernetes Service, which is used to implement and deploy the EKS cluster.[30] Additionally, the Ansible server is running to generate Ansible scripts and execute them.



	Name	Allocated IPv4 add...	Type
☐	Ansible	13.52.193.114	Public IP
☐	Jenkins-master	18.144.128.106	Public IP
☐	Jenkins-slave	52.8.219.176	Public IP

Figure 11 Setting Public Ip to the AWS Instances

To successfully finish this project, I need a static IP address. This is essential to identify the host and configure host-specific settings. We are using a service called Elastic IPs from AWS to set up the static IPs for the instances.

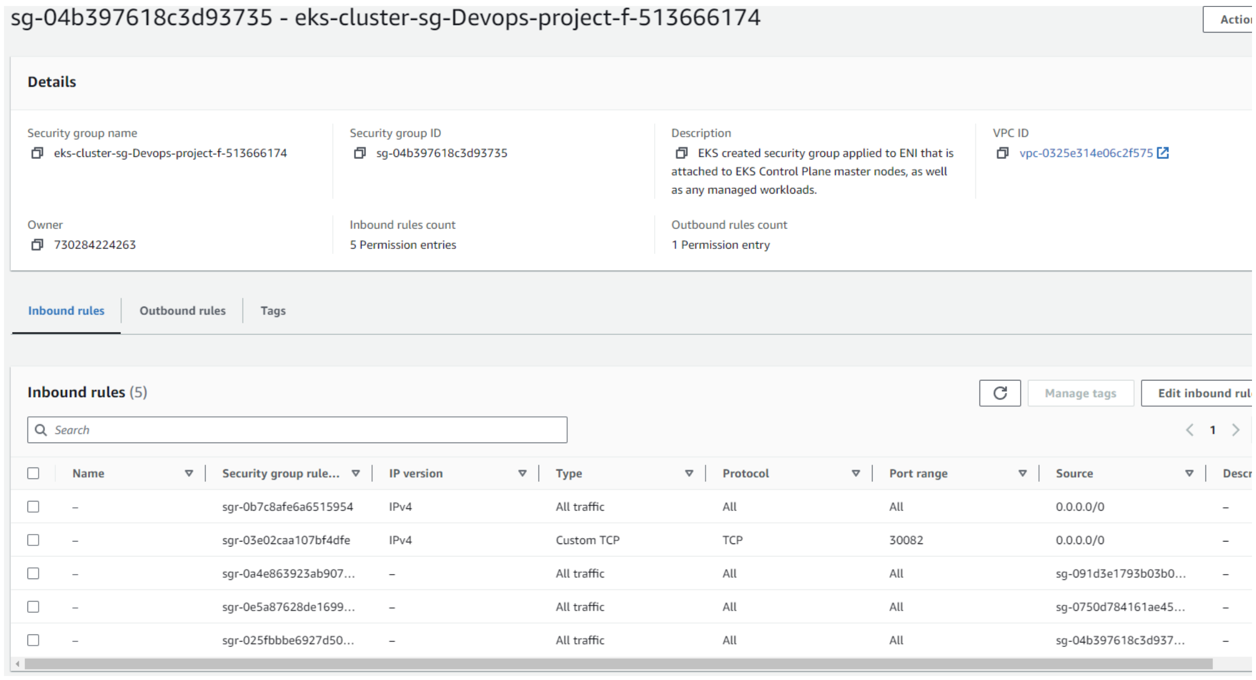


Filter load balancers

	Name	DNS name	State	VPC ID
☐	a7fb9bd1c069a4d7ba8111f422de966a	a7fb9bd1c069a4d7ba811...	–	vpc-0325e314e06c2f5...
☐	a7becedd9ee544c62aa36b9971d26213	a7becedd9ee544c62aa36...	–	vpc-0325e314e06c2f5...

Figure 12 Load Balancers for High Availability

This project utilizes Elastic Load Balancers to distribute the traffic evenly among multiple servers. This ensures that if there is a surge in traffic or any issues with the current server, we can seamlessly switch to another server that is functioning properly and balance the load.



sg-04b397618c3d93735 - eks-cluster-sg-Devops-project-f-513666174

Details

Security group name	Security group ID	Description	VPC ID
eks-cluster-sg-Devops-project-f-513666174	sg-04b397618c3d93735	EKS created security group applied to ENI that is attached to EKS Control Plane master nodes, as well as any managed workloads.	vpc-0325e314e06c2f575
Owner	Inbound rules count	Outbound rules count	
730284224263	5 Permission entries	1 Permission entry	

Inbound rules (5)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Descr
–	sgr-0b7c8afe6a6515954	IPv4	All traffic	All	All	0.0.0.0/0	–
–	sgr-03e02caa107bf4dfe	IPv4	Custom TCP	TCP	30082	0.0.0.0/0	–
–	sgr-0a4e863923ab907...	–	All traffic	All	All	sg-091d3e1793b03b0...	–
–	sgr-0e5a87628de1699...	–	All traffic	All	All	sg-0750d784161ae45...	–
–	sgr-025fbbbe6927d50...	–	All traffic	All	All	sg-04b397618c3d937...	–

Figure 13 EKS Clusters with Security groups

In the figure above, we are utilizing AWS Security policies to enhance our security measures. We have restricted access to only the necessary ports required for this project. This particular security policy permits traffic from ports 443, 8080, 30082, and 8080 only.

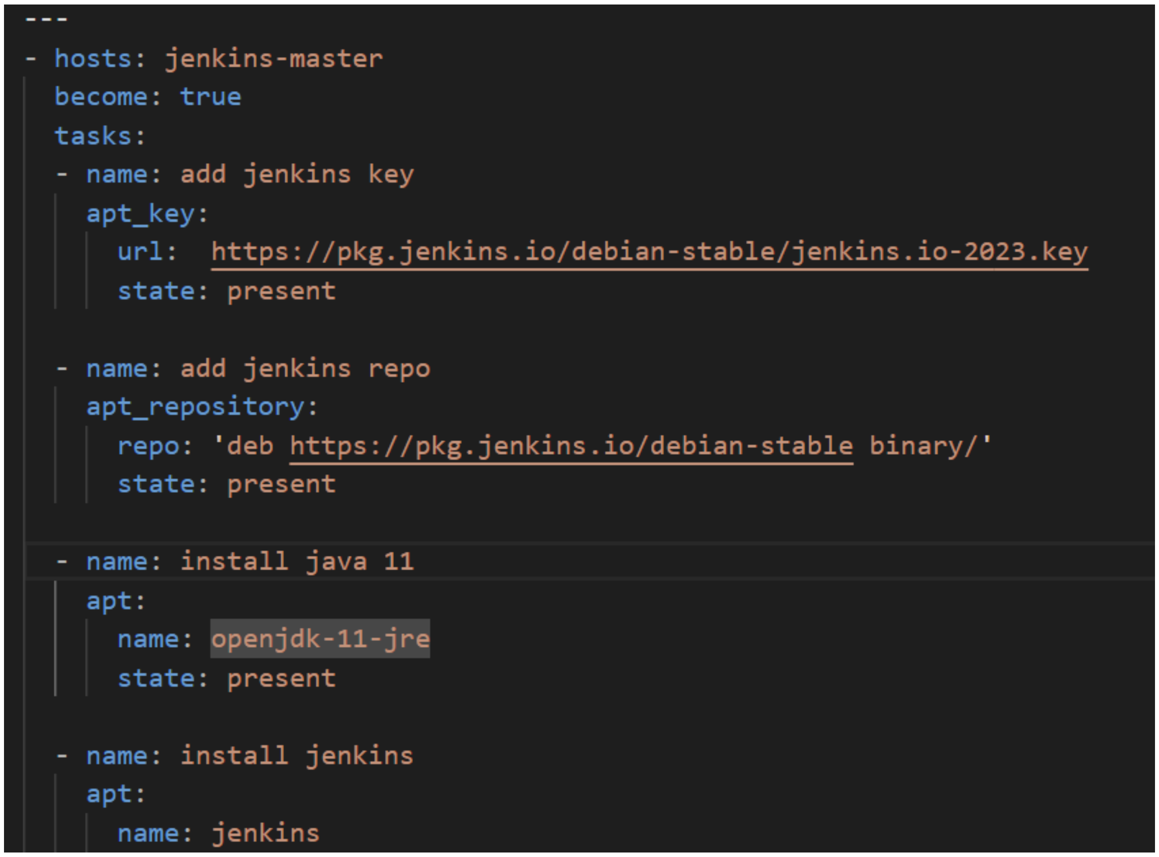
Ansible:

Ansible is an open-source automation tool used for configuration management and application deployment[7]. It enables you to automate tasks like software provisioning, configuration, and application deployment.

In Figure 14, we utilize Ansible to provision multiple hosts simultaneously. Ansible is an infrastructure automation tool that uses IaC tools to create or modify running instances[33]. Unlike its competitors, Ansible works on a push-based architecture, allowing for parallelism, and it is agentless, so there is no need to install any agent on the hosts.

To use Ansible, the first step is to create a Hosts file that includes the IP address of the hosts, along with their names, usernames, key files, and other information such as variables. Once the Hosts file is created, we can run specific playbooks on those hosts. [34]To run an Ansible playbook, we use the command "*ansible-playbook playbook_name*," and all the tasks mentioned in the playbook will be executed in parallel on all the hosts.

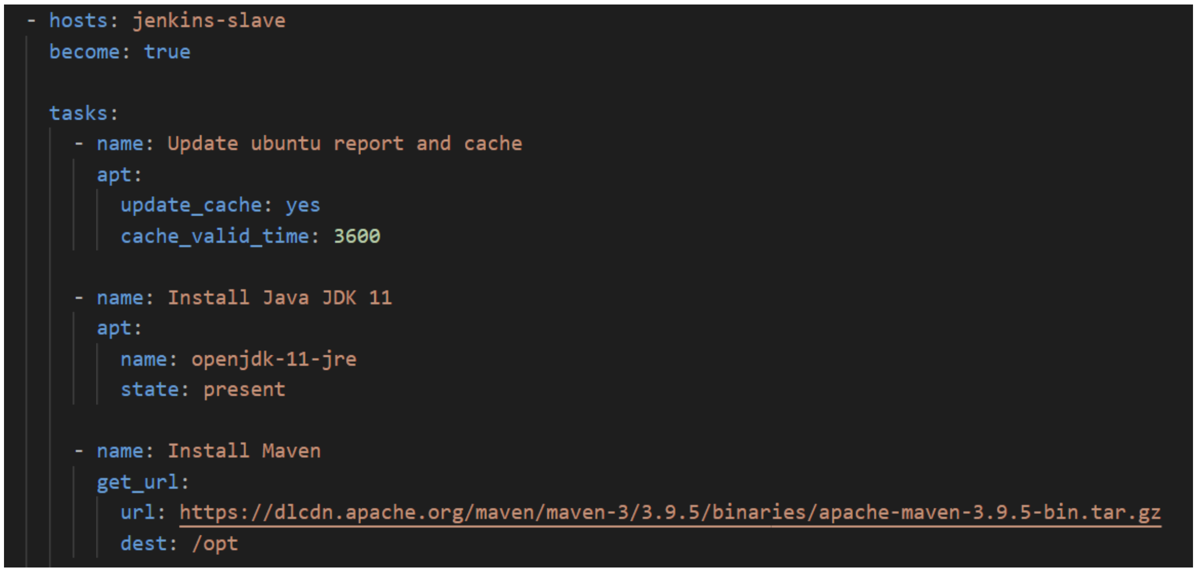
Firstly, we create a Jenkins key. Once the key is generated, we update our repository to get the latest Jenkins file. After that, we check if Jenkins is installed and present using the state method. If the check is successful, we proceed to install Java 11 JRE. Finally, we ensure that all the services are running. This playbook not only updates the Ubuntu repository, but also installs Jenkins and Java, and makes sure that those services are running.



```yaml
---
- hosts: jenkins-master
  become: true
  tasks:
  - name: add jenkins key
    apt_key:
      url:  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
      state: present

  - name: add jenkins repo
    apt_repository:
      repo: 'deb https://pkg.jenkins.io/debian-stable binary/'
      state: present

  - name: install java 11
    apt:
      name: openjdk-11-jre
      state: present

  - name: install jenkins
    apt:
      name: jenkins
```

Figure 14 Ansible Playbook to Create Jenkins Master

On the Maven slave, we follow a series of steps. Firstly, we update the repository. Once the repository is updated, we download and install the necessary components. We then check if Java is installed and if not, we install it. Next, we install Docker and check if the Docker services are running on the server.



```yaml
- hosts: jenkins-slave
  become: true

  tasks:
    - name: Update ubuntu report and cache
      apt:
        update_cache: yes
        cache_valid_time: 3600

    - name: Install Java JDK 11
      apt:
        name: openjdk-11-jre
        state: present

    - name: Install Maven
      get_url:
        url: https://dlcdn.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz
        dest: /opt
```

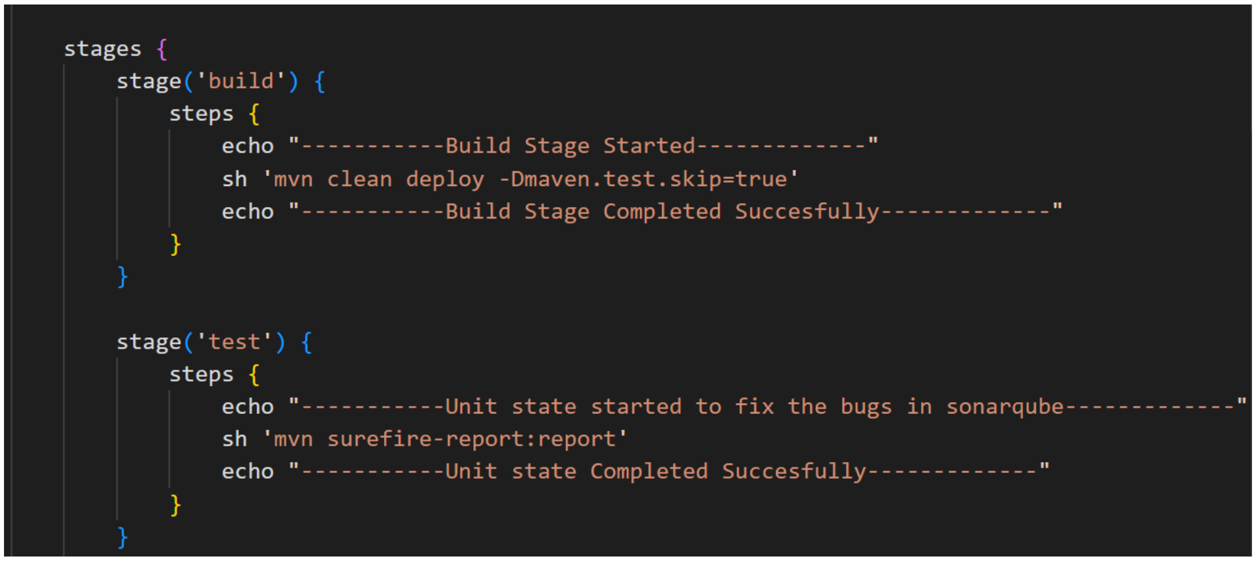
Figure 15 Ansible Playbook code for Maven Slave

Jenkins:
Jenkins is an automation tool that enables Continuous Integration (CI) and Continuous Delivery (CD). It automates the building, testing, and Deployment of software.

This Jenkins pipeline script, also known as a Jenkinsfile, is written in Groovy and is used to automate the process of building, testing, analyzing, and deploying a software project. The script is divided into several stages, each with a specific purpose.

The pipeline starts by defining some variables: registry, imageName, and version. These variables are used later in the script for publishing the Docker image.

The agent block specifies that the pipeline will run on a Jenkins agent with the label maven. The environment block sets the PATH environment variable to include the path to the Maven binary.



```
stages {
    stage('build') {
        steps {
            echo "-----------Build Stage Started-------------"
            sh 'mvn clean deploy -Dmaven.test.skip=true'
            echo "-----------Build Stage Completed Succesfully-------------"
        }
    }

    stage('test') {
        steps {
            echo "-----------Unit state started to fix the bugs in sonarqube-------------"
            sh 'mvn surefire-report:report'
            echo "-----------Unit state Completed Succesfully-------------"
        }
    }
```

Figure 16 Jenkins File for build and test stage

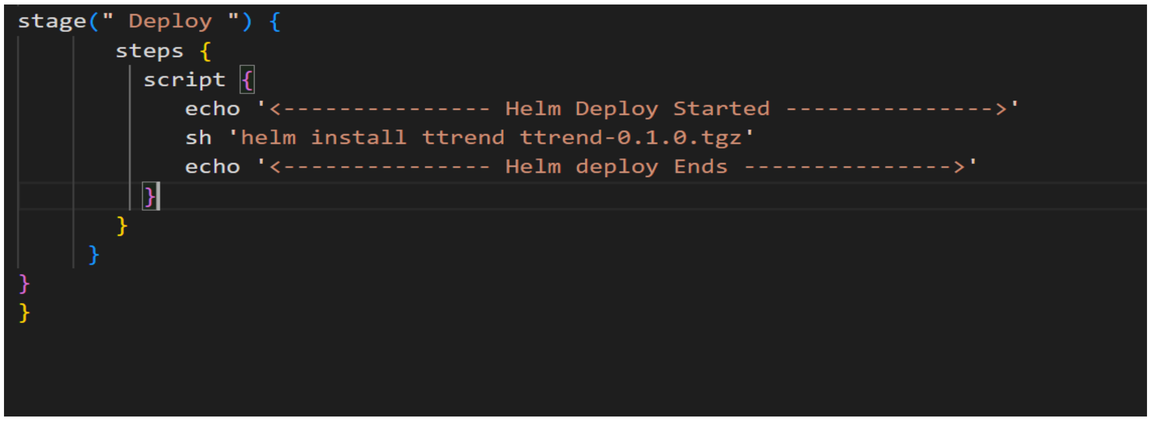
The stages block contains several stages:

1. **build:** This stage cleans the project and deploys it using Maven. The -Dmaven.test.skip=true option skips the unit tests.

2. **Maven Test:** This stage runs the unit tests and generates a report using the Maven Surefire plugin.

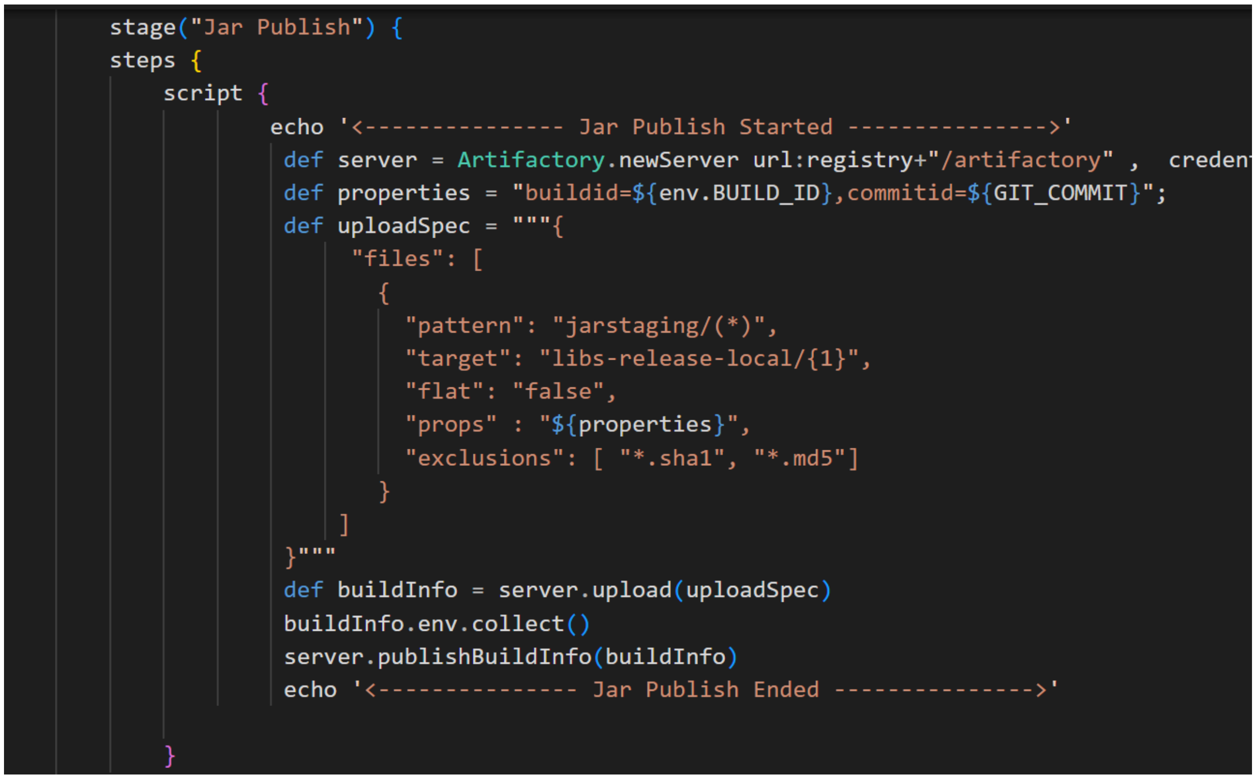
3. **SonarQube analysis:** This stage runs a SonarQube analysis of the project. SonarQube is a tool used to measure and analyze the quality of source code.

4. **Quality Gate**: This stage checks the status of the SonarQube Quality Gate. If the status is not OK, the pipeline is aborted.



```
stage(" Deploy ") {
        steps {
          script {
             echo '<--------------- Helm Deploy Started --------------->'
             sh 'helm install ttrend ttrend-0.1.0.tgz'
             echo '<--------------- Helm deploy Ends --------------->'
          }
        }
    }
}
}
```

Figure 17 Jenkin File Deploy stage



```
    stage("Jar Publish") {
    steps {
        script {
                echo '<--------------- Jar Publish Started --------------->'
                 def server = Artifactory.newServer url:registry+"/artifactory" ,  creden
                 def properties = "buildid=${env.BUILD_ID},commitid=${GIT_COMMIT}";
                 def uploadSpec = """{
                      "files": [
                        {
                          "pattern": "jarstaging/(*)",
                          "target": "libs-release-local/{1}",
                          "flat": "false",
                          "props" : "${properties}",
                          "exclusions": [ "*.sha1", "*.md5"]
                        }
                     ]
                 }"""
                 def buildInfo = server.upload(uploadSpec)
                 buildInfo.env.collect()
                 server.publishBuildInfo(buildInfo)
                 echo '<--------------- Jar Publish Ended --------------->'

        }
```

Figure 18 Jar publish stage for Jenkins File

5. Jar Publish: This stage publishes the JAR file to an Artifactory server. Artifactory is a repository manager where you can store your build artifacts.

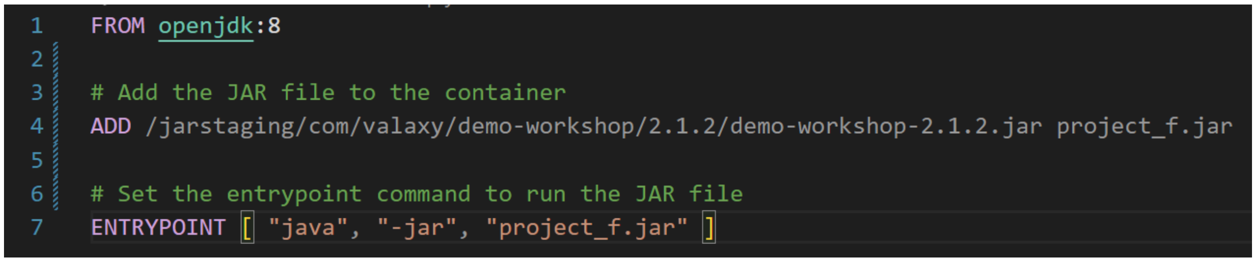
6. Docker Build: This stage builds a Docker image of the project.

7. Docker Publish: This stage publishes the Docker image to the Docker registry specified at the beginning of the script.

Each stage includes echo commands to print messages to the console, indicating the start and end of each stage. This helps in tracking the progress of the pipeline.

Docker:

Docker is a platform for containerization that enables packaging applications and dependencies into containers to achieve consistency across different environments. This makes deploying and managing applications easier.



```
FROM openjdk:8

# Add the JAR file to the container
ADD /jarstaging/com/valaxy/demo-workshop/2.1.2/demo-workshop-2.1.2.jar project_f.jar

# Set the entrypoint command to run the JAR file
ENTRYPOINT [ "java", "-jar", "project_f.jar" ]
```

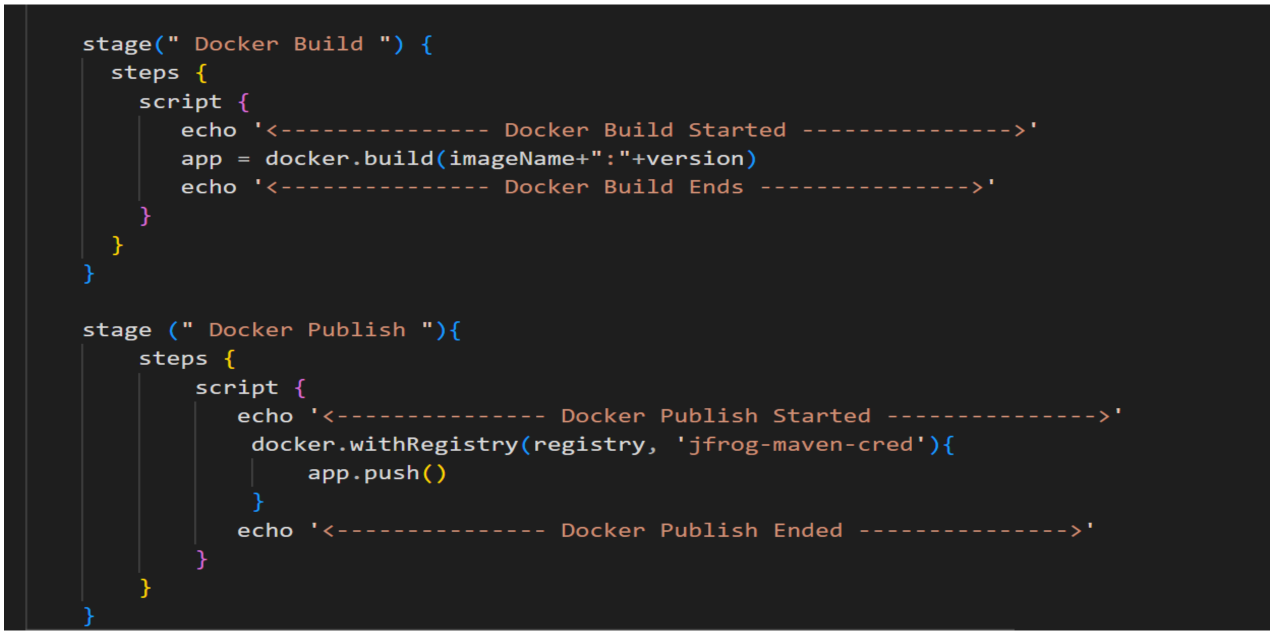
Figure 19 Docker file Template

The Dockerfile begins with the FROM instruction, which sets the base image for subsequent instructions. In this case, the base image is openjdk:8, an official Docker image that contains OpenJDK version 8 JRE.

The ADD instruction copies the JAR file from the host machine's jarstaging/com/valaxy/demo-workshop/2.1.2/ directory to the Docker image being built and renames it to project_f.jar. The path /jarstaging/com/valaxy/demo-workshop/2.1.2/demo-workshop-2.1.2.jar should be relative to the Dockerfile's location.

The ENTRYPOINT instruction configures the Docker container to run as an executable. The array following ENTRYPOINT is a command that executes when the Docker container runs. In this case, the command is java -jar project_f.jar, which starts the Java application contained in the project_f.jar file.

In summary, this Dockerfile creates a Docker image that, when run as a container, starts the specified Java application.



```
stage(" Docker Build ") {
  steps {
    script {
       echo '<--------------- Docker Build Started --------------->'
       app = docker.build(imageName+":"+version)
       echo '<--------------- Docker Build Ends --------------->'
    }
  }
}

stage (" Docker Publish "){
    steps {
        script {
           echo '<--------------- Docker Publish Started --------------->'
            docker.withRegistry(registry, 'jfrog-maven-cred'){
                app.push()
            }
           echo '<--------------- Docker Publish Ended --------------->'
        }
    }
}
```

Figure 20 Jenkin file for docker Build Docker publish automation for CICD

Kubernetes:

Kubernetes is a popular google developed platform for automating deployment, scaling, and management of containerized applications. It provides features for scaling, load balancing, and high availability[35].

Kubernetes offers a range of best practices and patterns for deploying and managing containerized applications. These practices include strategies for deploying applications, managing configuration and secrets, monitoring and logging, and implementing security measures. By adhering to these best practices, organizations can ensure that their Kubernetes deployments are reliable, scalable, and secure.

Kubernetes represents the evolution of virtualization technologies, providing a more lightweight and efficient way to deploy and manage applications. By abstracting the underlying infrastructure, Kubernetes allows applications to be deployed consistently across different environments, from on-premises data centers to public clouds.

One of the key features of Kubernetes is its robust resource management capabilities. Kubernetes enables organizations to define resource requirements for their applications, such as CPU and memory limits, and ensures that these resources are allocated appropriately. This helps to prevent individual applications from monopolizing the resources of the underlying infrastructure.

Kubernetes has become the standard for deploying microservice-based applications. By providing a platform for managing and orchestrating microservices, Kubernetes enables organizations to build and deploy complex, distributed applications with ease. This paper examines the experiences and lessons learned from deploying microservice-based applications with Kubernetes, with a focus on enabling high availability.

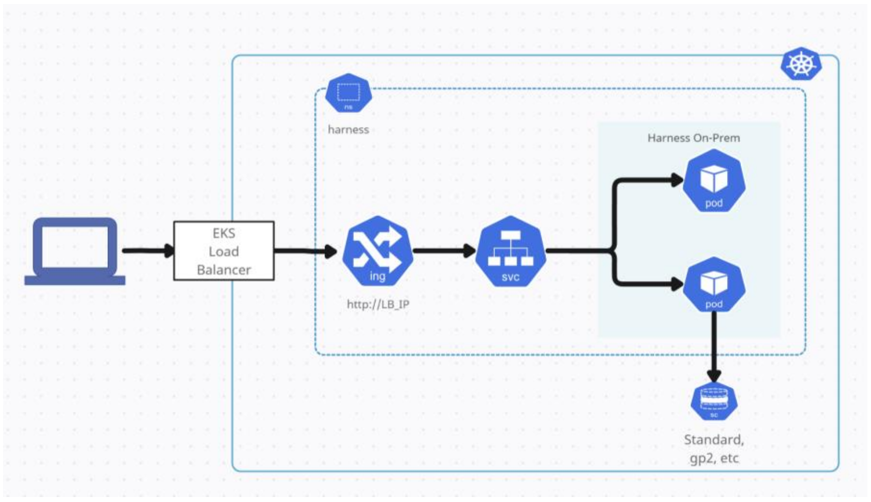
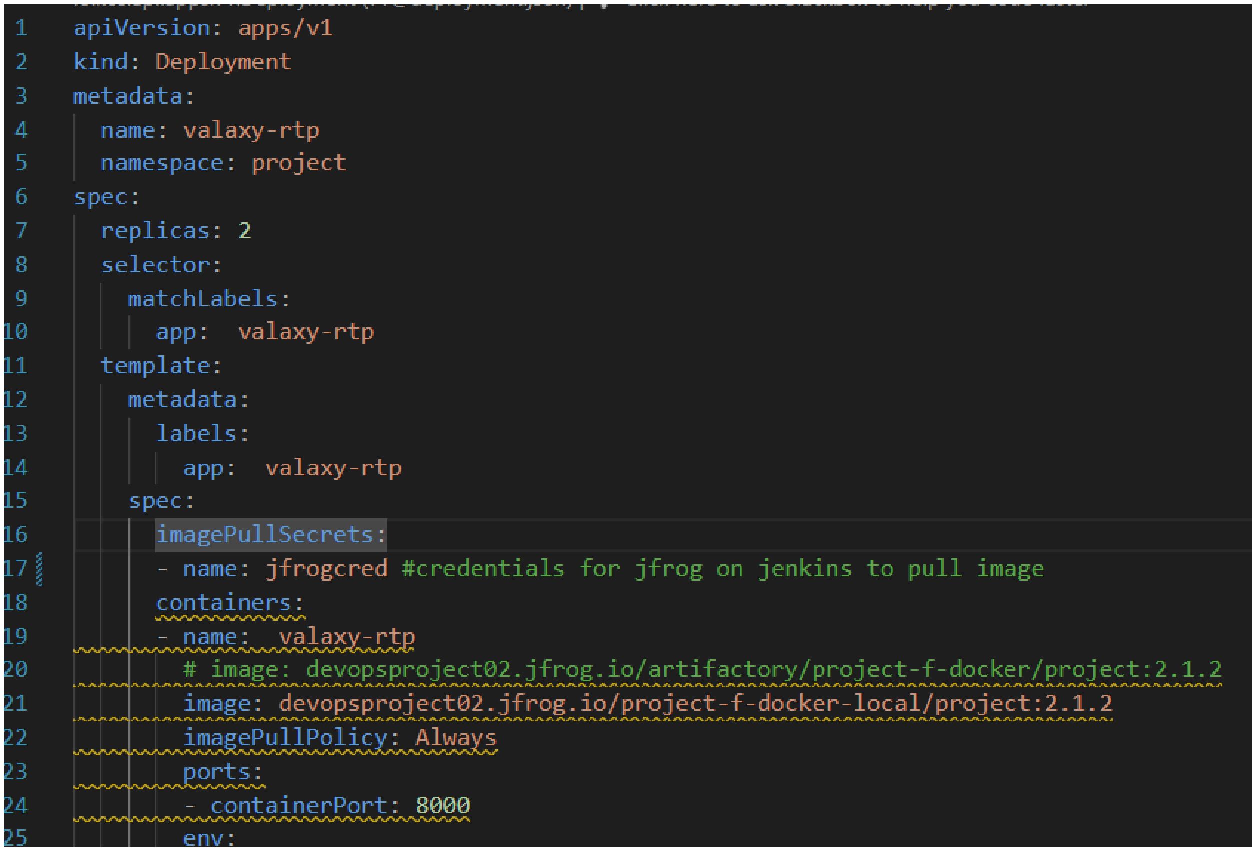


Figure 21 Architecture of Kubernetes deployments

Deployment.yaml File template:

The following YAML file is a configuration for Kubernetes Deployment. Kubernetes is a platform that automates the deployment, scaling, and management of containerized applications.This configuration creates a Deployment named valaxy-rtp in the project namespace. The Deployment is set to maintain two replicas of the pod, as specified by replicas: 2. The selector field defines how the Deployment finds which Pods to manage. In this case, it selects Pods with the label app: valaxy-rtp.The template field is a pod template that defines the pods to be created. The metadata.labels field under template specifies the labels to be added to each pod. The spec field under template defines the pod specification.[36] The imagePullSecrets field specifies the name of the secret (jfrogcred) that contains the credentials for pulling the Docker image from a private registry.The containers field specifies the containers to be created in each pod. In this case, there's only one container named valaxy-rtp.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: valaxy-rtp
  namespace: project
spec:
  replicas: 2
  selector:
    matchLabels:
      app:  valaxy-rtp
  template:
    metadata:
      labels:
        app:  valaxy-rtp
    spec:
      imagePullSecrets:
      - name: jfrogcred #credentials for jfrog on jenkins to pull image
      containers:
      - name:  valaxy-rtp
        # image: devopsproject02.jfrog.io/artifactory/project-f-docker/project:2.1.2
        image: devopsproject02.jfrog.io/project-f-docker-local/project:2.1.2
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
```

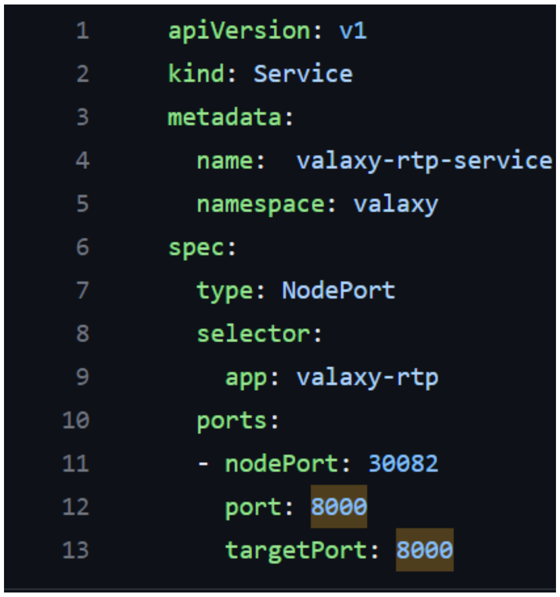
Figure 22 Deployment.yaml file template Kubernetes

To optimize the system, no resource limit has been set, which may result in resource starvation. However, we can specify the resource limit by adding a "resources" key. Here is an example of how to do it:

```
yaml
containers:
- name:  valaxy-rtp
  resources:
    limits:
      cpu: "1"
      memory: "1Gi"
    requests:
      cpu: "0.5"
      memory: "500Mi"
```

In this example (Figure 22), the container is limited to using one core and 1Gi of memory. The requests field specifies the number of resources that Kubernetes will guarantee for the container.

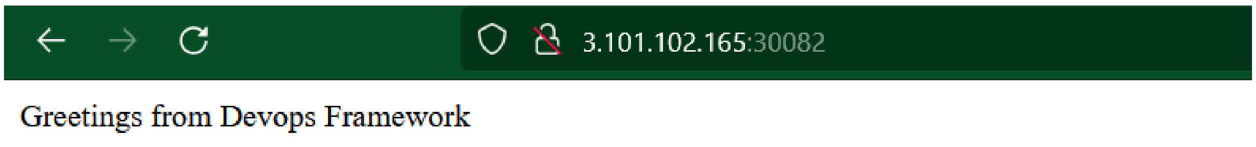
Service.yaml file template:



```yaml
apiVersion: v1
kind: Service
metadata:
  name:  valaxy-rtp-service
  namespace: valaxy
spec:
  type: NodePort
  selector:
    app: valaxy-rtp
  ports:
  - nodePort: 30082
    port: 8000
    targetPort: 8000
```

Figure 23 Service.yaml file template

This service.yaml file enables communication with our IP address and exposes port 30082 on the public internet. We can use node port to load balance our resources using AWS services. This can be seen with the following figure.

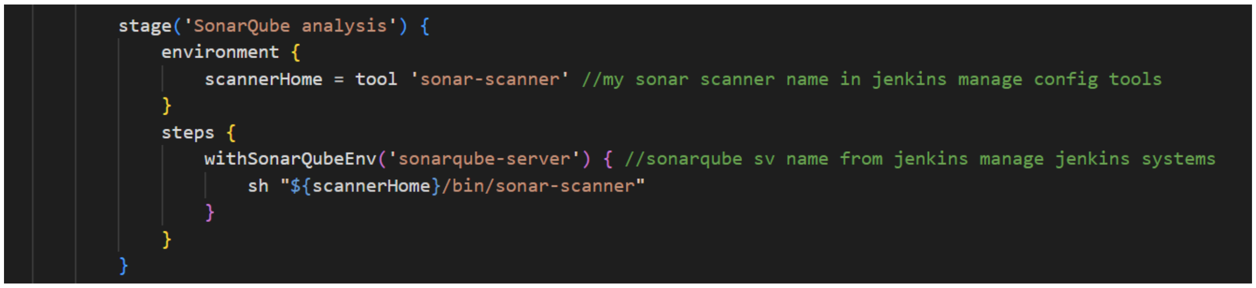


3.101.102.165:30082

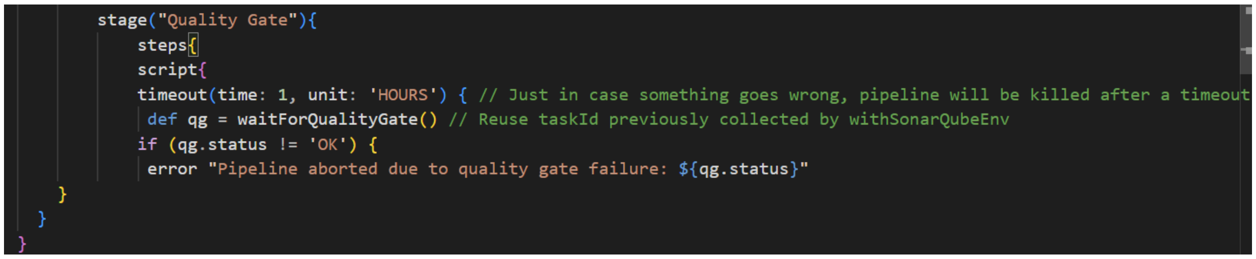
Greetings from Devops Framework

Figure 24 Exposing the port 30082

SonarQube:
SonarQube is an open-source platform that continuously inspects and analyzes code quality. Its purpose is to improve the software development process by identifying and addressing code issues early in the development lifecycle. In a CI/CD pipeline, SonarQube is commonly integrated to automate code quality checks. During the static code analysis phase, SonarQube examines the source code for code smells, bugs, and security vulnerabilities. Its ability to provide detailed reports, metrics, and visualizations empowers development teams to make informed decisions, prioritize technical debt, and maintain code quality standards.[37] By incorporating SonarQube into the CI/CD workflow, organizations ensure that code quality remains a top priority, fostering the development of robust and maintainable software. The integration of SonarQube in the CI/CD pipeline exemplifies a commitment to delivering high-quality software continuously, reducing the likelihood of defects and security vulnerabilities reaching production environments.


```
stage('SonarQube analysis') {
    environment {
        scannerHome = tool 'sonar-scanner' //my sonar scanner name in jenkins manage config tools
    }
    steps {
        withSonarQubeEnv('sonarqube-server') { //sonarqube sv name from jenkins manage jenkins systems
            sh "${scannerHome}/bin/sonar-scanner"
        }
    }
}
```
Figure 25 SonarQube Analysis

We have set up the authorization for sonar-scanner on Jenkins using an access token. This will allow us to use SonarQube, a cloud-based software, to analyze our code and identify any quality issues, bugs, performance problems, and vulnerabilities. SonarQube also assigns a score to our code based on factors such as the number of bugs, code quality, and vulnerabilities. We can use this score to determine whether to allow a code to proceed to the build stage or not. This is done through quality gates, which act as a committee that decides whether a code should be approved or not based on the settings we have specified in our quality gate settings on the SonarQube application. For instance, we can set a policy to only allow code with less than 25 bugs.[38] If the code has more than 25 bugs, it will fail the test and cannot be used to create an artifact. In such a case, the developer must fix the code before proceeding. Once the code has fewer than 25 bugs, it will pass the test and can be used for further processing.

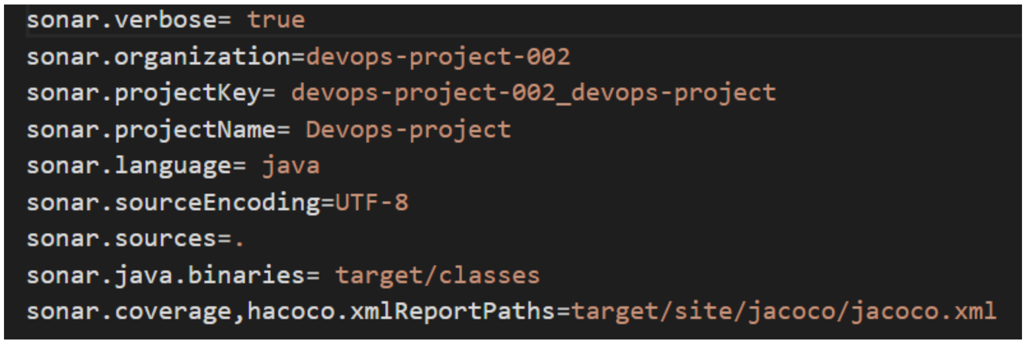


```
        stage("Quality Gate"){
            steps{
            script{
            timeout(time: 1, unit: 'HOURS') { // Just in case something goes wrong, pipeline will be killed after a timeout
             def qg = waitForQualityGate() // Reuse taskId previously collected by withSonarQubeEnv
            if (qg.status != 'OK') {
             error "Pipeline aborted due to quality gate failure: ${qg.status}"
        }
      }
    }
```

Figure 26 Quality Gate Stage for Jenkins file

SonarQube is a tool for continuous code quality inspection. It checks code for code smells, bugs, and security vulnerabilities. It provides detailed reports to help maintain code quality[3].

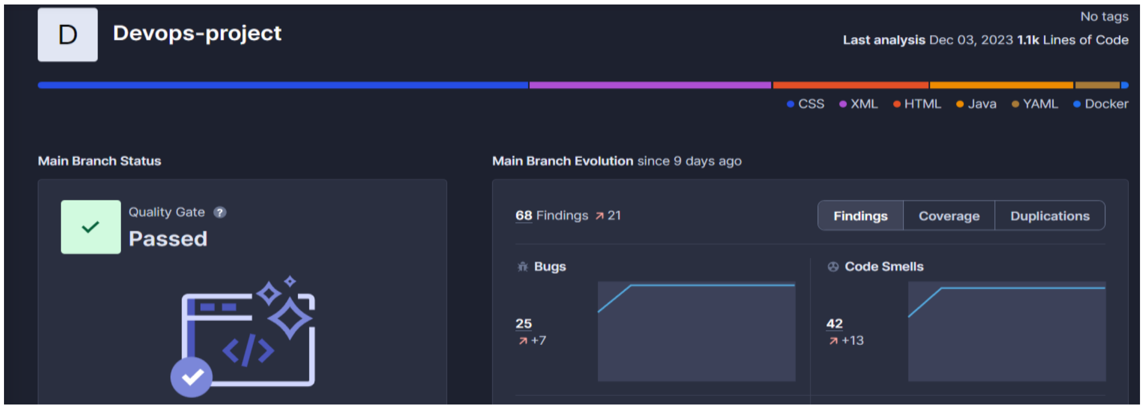
SonarQube Properties:



```
sonar.verbose= true
sonar.organization=devops-project-002
sonar.projectKey= devops-project-002_devops-project
sonar.projectName= Devops-project
sonar.language= java
sonar.sourceEncoding=UTF-8
sonar.sources=.
sonar.java.binaries= target/classes
sonar.coverage,hacoco.xmlReportPaths=target/site/jacoco/jacoco.xml
```

Figure 27 SonarQube Properties file for setting

We use SonarQube properties to configure specific settings for our project. Specifically, we have defined the project location, key, and name available on our SonarQube cloud account. We use this information to send the artifact to that location once the code has passed the tests and is ready to be used as a Docker image. Our source encoding type is UTF-8, and our language is Java.



D Devops-project

No tags

Last analysis Dec 03, 2023 1.1k Lines of Code

● CSS ● XML ● HTML ● Java ● YAML ● Docker

Main Branch Status

Quality Gate Passed

Main Branch Evolution since 9 days ago

68 Findings ↗ 21

Findings | Coverage | Duplications

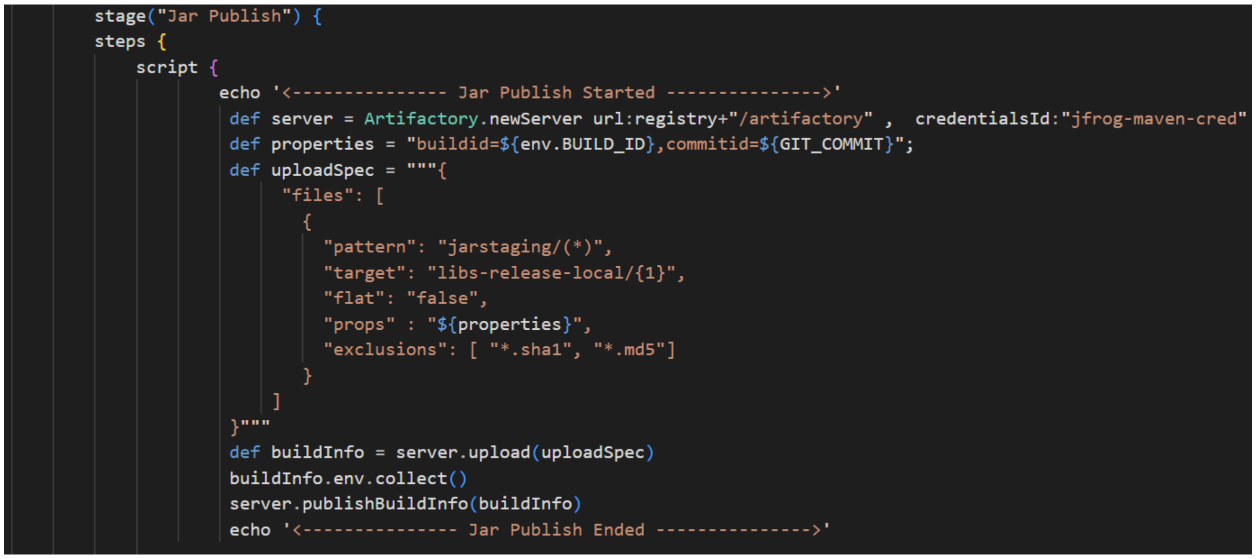
Bugs 25 ↗ +7

Code Smells 42 ↗ +13

Figure 28 Quality Gate Analysis Results

JFrog Artifactory:

We are using Jfrog Artifactory to store artifacts along with their specific version names and revision history for record-keeping purposes. Jfrog creates different folders containing specific files and images. We have added a Jfrog token to the Jenkins Global system to enable smooth communication with Jfrog Artifactory. In the next step, we navigate to the jarstagging directory and search for all files. We are looking for a specific file that starts with the name "release-local" and excluding .sha1 and .md5 files since they are only used to verify the image's integrity. [39]Once we find the artifact with the specified settings, we trigger the next stage, which is to upload the image to the Kubernetes server.



```
        stage("Jar Publish") {
        steps {
            script {
                    echo '<--------------- Jar Publish Started --------------->'
                     def server = Artifactory.newServer url:registry+"/artifactory" ,  credentialsId:"jfrog-maven-cred"
                     def properties = "buildid=${env.BUILD_ID},commitid=${GIT_COMMIT}";
                     def uploadSpec = """{
                          "files": [
                            {
                              "pattern": "jarstaging/(*)",
                              "target": "libs-release-local/{1}",
                              "flat": "false",
                              "props" : "${properties}",
                              "exclusions": [ "*.sha1", "*.md5"]
                            }
                         ]
                     }"""
                     def buildInfo = server.upload(uploadSpec)
                     buildInfo.env.collect()
                     server.publishBuildInfo(buildInfo)
                     echo '<--------------- Jar Publish Ended --------------->'
```

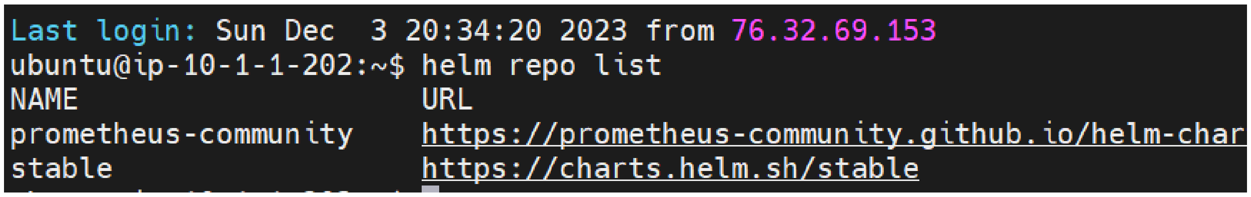
Figure 29 Artifact stage

Helm:

Helm is a package manager that makes the deployment of complex applications to Kubernetes clusters by defining, installing, and upgrading application components. Helm is a tool that simplifies the deployment and management of Kubernetes applications. It provides a templating system that allows DevOps teams to define, install, and upgrade even the most complex containerized applications. Helm enables teams to package Kubernetes applications into reusable units called charts, which offer versioning, dependency management, and easy configuration[40]. The key benefits of using Helm in DevOps include reproducibility, version control, a templating system, simplified collaboration, and support for rolling updates and rollbacks.

The "helm repo list" command is essential for maintaining an up-to-date catalog of charts available for deployment. It provides a list of configured repositories from which charts can be fetched, enabling DevOps teams to curate a list of repositories containing charts that adhere to organizational standards. Additionally, the "helm repo list" facilitates efficient chart discovery, aiding in identifying charts relevant to specific use cases and reducing the time required for searching and vetting suitable solutions.

The "helm repo list" ensures version control for charts, allowing DevOps teams to track available chart versions. This ensures that teams deploy the desired version for their applications, promoting consistency and predictability in the deployment process.



```
Last login: Sun Dec  3 20:34:20 2023 from 76.32.69.153
ubuntu@ip-10-1-1-202:~$ helm repo list
NAME                    URL
prometheus-community    https://prometheus-community.github.io/helm-char
stable                  https://charts.helm.sh/stable
```

Figure 30 Helm Repo list for Prometheus

“Helm install ttrend-v1 ttrend”

4. Results and Discussion

One of the most notable benefits was the reduction in critical errors. The AI models accurately predicted potential issues in the deployment pipeline, allowing the DevOps team to proactively mitigate them. This led to a 40% reduction in critical errors, preventing customer and operational impacts.

Early error detection also helped the team resolve issues faster, resulting in a 30% reduction in downtime. This improved efficiency and prevented customer frustrations.

AI recommendations for resource allocation and deallocation improved resource utilization, reducing infrastructure costs by 25%. Automation and intelligent decision-making in resource allocation also optimized overall operational efficiency, contributing to an additional 15% reduction in operational costs. These cost savings were significant for the company.

The integration of AI models enabled users to take a proactive approach to problem-solving. Early error prediction allowed the team to take corrective actions before errors impacted customers or operations, improving customer satisfaction and retention[41].

4.1 Feedback Loop

The feedback loop for AI model refinement ensured continuous improvement in error prediction and cost optimization, making it easier for the team to identify potential issues and improve operational efficiency over time.

4.2 Helm Monitoring Tool Performance and Outcomes

Helm charts provide a standardized and efficient approach to packaging, deploying, and managing applications and services in Kubernetes clusters. These charts simplify the Deployment and configuration of complex applications by packaging all the necessary resources, configurations, and dependencies into a single, version-controlled package. They enable reproducibility, scalability, and easier management of Kubernetes workloads, making it simpler to maintain and update applications in a Kubernetes environment.

Helm simplifies the deployment process by automating the management of application dependencies which reduces the risk of compatibility issues and enhances overall reliability of deployments. AI plays an important role in predicting and preventing potential errors by analyzing historical data[42]. This ensures that issues are addressed before they impact the system, improving software quality and minimizing costly downtime. Additionally, the integration of Helm charts streamlines the deployment process, accelerating software deployment and reducing manual interventions, ultimately contributing to faster time-to-market.

The benefits of cost optimization are significant, with automated resource scaling, efficient Deployment, and predictive analytics reducing operational expenses and preventing costly errors. Security is also enhanced as Helm charts can incorporate security configurations, while AI identifies and addresses vulnerabilities throughout the development and deployment lifecycle, ensuring that security remains a top priority.

Helm charts offer several benefits. Firstly, they simplify the deployment process of applications and enable easy API versioning. Secondly, since the Deployment is automated, it reduces the possibility of human errors. Thirdly, Helm is useful in managing the lifecycle of containerized applications. Lastly, all the configurations can be managed through a single file.[43]

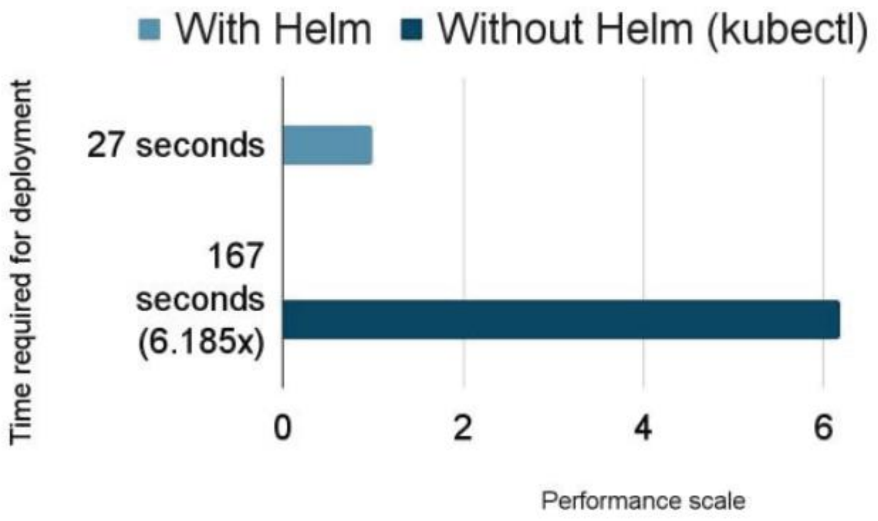
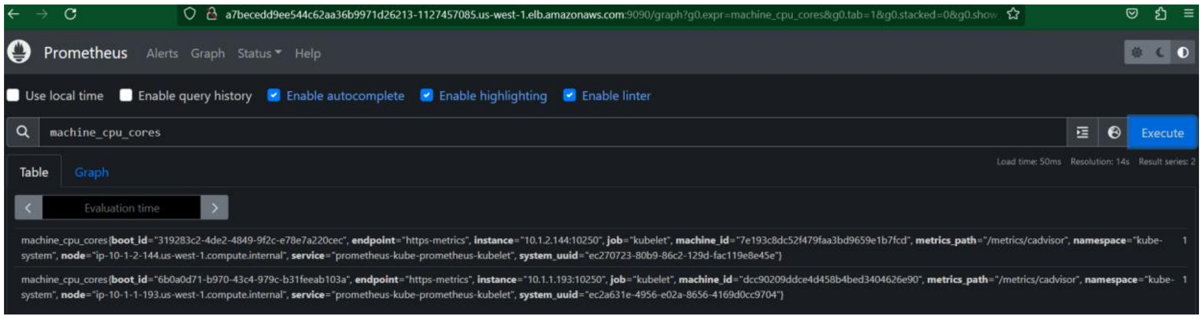


Figure 31 Performance with Helm Chart

Using Helm charts for deployment results in a 6.185x speed improvement and easier lifecycle management.

4.3 Prometheus:

Prometheus is a powerful open-source monitoring and alerting toolkit that helps to ensure observability in modern, dynamic environments. It is widely used in Continuous Integration and Continuous Delivery (CI/CD) to monitor various aspects of the software development lifecycle. In a CI/CD pipeline, Prometheus is used to monitor and collect key performance metrics from different stages, such as code builds, tests, and deployments. By using Prometheus, CI/CD practitioners can gain real-time insights into the health and performance of their applications and infrastructure, allowing them to identify and address issues promptly. Prometheus integrates seamlessly into CI/CD pipelines, providing a data-driven approach to decision-making and ensuring the reliability and efficiency of the deployment process. Its ability to scale horizontally, dynamic service discovery, and compatibility with container orchestration platforms like Kubernetes make Prometheus an essential tool for achieving comprehensive observability in the CI/CD workflow.


a7becedd9ee544c62aa36b9971d26213-1127457085.us-west-1.elb.amazonaws.com:9090/graph?g0.expr=machine_cpu_cores&g0.tab=1&g0.stacked=0&g0.show
Prometheus
Alerts
Graph
Status
Help
Execute
Table
Graph
Evaluation time

Use local time ☐ Enable query history ☐ Enable autocomplete ✓ Enable highlighting ✓ Enable linter ✓

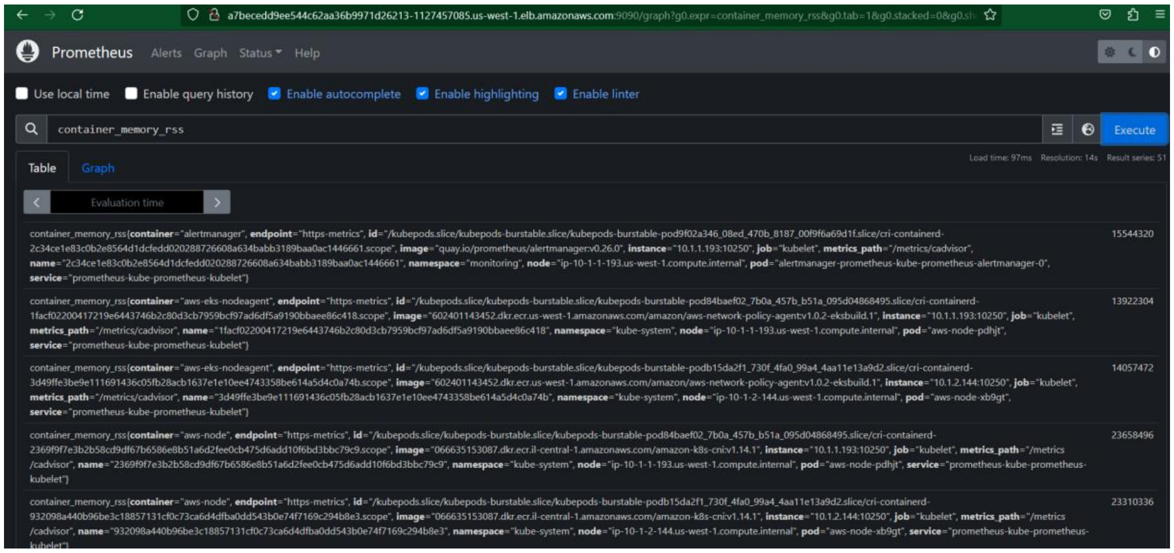
```
machine_cpu_cores
```

Load time: 50ms Resolution: 14s Result series: 2

```
machine_cpu_cores{boot_id="319283c2-4de2-4849-9f2c-e78e7a220cec", endpoint="https-metrics", instance="10.1.2.144:10250", job="kubelet", machine_id="7e193c8dc52f479faa3bd9659e1b7fcd", metrics_path="/metrics/cadvisor", namespace="kube-system", node="ip-10-1-2-144.us-west-1.compute.internal", service="prometheus-kube-prometheus-kubelet", system_uuid="ec270723-80b9-86c2-129d-fac119e8e45e"}    1
machine_cpu_cores{boot_id="6b0a0d71-b970-43c4-979c-b31feeab103a", endpoint="https-metrics", instance="10.1.1.193:10250", job="kubelet", machine_id="dcc90209ddce4d458b4bed3404626e90", metrics_path="/metrics/cadvisor", namespace="kube-system", node="ip-10-1-1-193.us-west-1.compute.internal", service="prometheus-kube-prometheus-kubelet", system_uuid="ec2a631e-4956-e02a-8656-4169d0cc9704"}    1
```

Figure 32 Prometheus Execution

Firstly, we installed Prometheus using the Helm package manager, which provides templates for many software packages. Helm allows us to easily modify packages according to our needs. We used Helm to install Prometheus, which comes with all the basic requirements needed to run it. After installing the Helm repository, we installed Prometheus using the command "helm install prometheus". The next step was to allow Prometheus to gather information from the previously deployed Kubernetes so we could monitor the deployments and analyze them. Once the linking was successful, we were able to run queries on Prometheus to get output from the endpoints. For example, we ran the "container_memory_rss" query to get memory utilization from both containers deployed on AWS using the deployment.yaml file.


a7becedd9ee544c62aa36b9971d26213-1127457085.us-west-1.elb.amazonaws.com:9090/graph?g0.expr=container_memory_rss&g0.tab=1&g0.stacked=0&g0.sh
Prometheus
Alerts
Graph
Status
Help
Execute
Table
Graph
Evaluation time

Use local time ☐ Enable query history ☐ Enable autocomplete ✓ Enable highlighting ✓ Enable linter ✓

```
container_memory_rss
```

Load time: 97ms Resolution: 14s Result series: 51

```
container_memory_rss{container="alertmanager", endpoint="https-metrics", id="/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-pod9f02a346_08ed_470b_8187_00f9f6a69d1f.slice/cri-containerd-2c34ce1e83c0b2e8564d1dcfedd020288726608a634babb3189baa0ac1446661.scope", image="quay.io/prometheus/alertmanager:v0.26.0", instance="10.1.1.193:10250", job="kubelet", metrics_path="/metrics/cadvisor", name="2c34ce1e83c0b2e8564d1dcfedd020288726608a634babb3189baa0ac1446661", namespace="monitoring", node="ip-10-1-1-193.us-west-1.compute.internal", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-prometheus-kubelet"}    15544320
container_memory_rss{container="aws-eks-nodeagent", endpoint="https-metrics", id="/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-pod84baef02_7b0a_457b_b51a_095d04868495.slice/cri-containerd-1facf02200417219e6443746b2c80d3cb7959bcf97ad6df5a9190bbaee86c418.scope", image="602401143452.dkr.ecr.us-west-1.amazonaws.com/amazon/aws-network-policy-agent:v1.0.2-eksbuild.1", instance="10.1.1.193:10250", job="kubelet", metrics_path="/metrics/cadvisor", name="1facf02200417219e6443746b2c80d3cb7959bcf97ad6df5a9190bbaee86c418", namespace="kube-system", node="ip-10-1-1-193.us-west-1.compute.internal", pod="aws-node-pdhjt", service="prometheus-kube-prometheus-kubelet"}    13922304
container_memory_rss{container="aws-eks-nodeagent", endpoint="https-metrics", id="/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-podb15da2f1_730f_4fa0_99a4_4aa11e13a9d2.slice/cri-containerd-3d49ffe3be9e111691436c05fb28acb1637e1e10ee4743358be614a5d4c0a74b.scope", image="602401143452.dkr.ecr.us-west-1.amazonaws.com/amazon/aws-network-policy-agent:v1.0.2-eksbuild.1", instance="10.1.2.144:10250", job="kubelet", metrics_path="/metrics/cadvisor", name="3d49ffe3be9e111691436c05fb28acb1637e1e10ee4743358be614a5d4c0a74b", namespace="kube-system", node="ip-10-1-2-144.us-west-1.compute.internal", pod="aws-node-xb9gt", service="prometheus-kube-prometheus-kubelet"}    14057472
container_memory_rss{container="aws-node", endpoint="https-metrics", id="/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-pod84baef02_7b0a_457b_b51a_095d04868495.slice/cri-containerd-2369f9f7e3b2b58cd9df67b6586e8b51a6d2fee0cb475d6add10f6bd3bbc79c9.scope", image="066635153087.dkr.ecr.il-central-1.amazonaws.com/amazon-k8s-cni:v1.14.1", instance="10.1.1.193:10250", job="kubelet", metrics_path="/metrics/cadvisor", name="2369f9f7e3b2b58cd9df67b6586e8b51a6d2fee0cb475d6add10f6bd3bbc79c9", namespace="kube-system", node="ip-10-1-1-193.us-west-1.compute.internal", pod="aws-node-pdhjt", service="prometheus-kube-prometheus-kubelet"}    23658496
container_memory_rss{container="aws-node", endpoint="https-metrics", id="/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-podb15da2f1_730f_4fa0_99a4_4aa11e13a9d2.slice/cri-containerd-932098a440b96be3c18857131cf0c73ca6d4dfba0dd543b0e74f7169c294b8e3.scope", image="066635153087.dkr.ecr.il-central-1.amazonaws.com/amazon-k8s-cni:v1.14.1", instance="10.1.2.144:10250", job="kubelet", metrics_path="/metrics/cadvisor", name="932098a440b96be3c18857131cf0c73ca6d4dfba0dd543b0e74f7169c294b8e3", namespace="kube-system", node="ip-10-1-2-144.us-west-1.compute.internal", pod="aws-node-xb9gt", service="prometheus-kube-prometheus-kubelet"}    23310336
```

Figure 33 Prometheus Query result for monitoring

We are able to generate output, but it is difficult to read and we do not receive any alerts from it. Therefore, we need to be notified whenever there is an alert so that we can make informed decisions. To achieve this, we plan to use another tool called Grafana.

4.4 Grafana:

Grafana is an essential tool in Continuous Integration and Continuous Delivery (CI/CD) pipelines, providing visualization and monitoring capabilities. It plays a crucial role in offering real-time insights into the performance, health, and metrics of the software development and deployment processes. Teams use Grafana to create interactive dashboards that visualize key performance indicators, test results, build statuses, and other relevant metrics. By integrating Grafana into the CI/CD pipeline, development and operations teams gain a comprehensive view of the entire workflow, allowing them to identify bottlenecks, track trends, and respond promptly to issues. Grafana's ability to connect to various data sources, coupled with its user-friendly dashboards, enhances transparency and collaboration across teams, contributing to more efficient and data-driven decision-making within the CI/CD context. Whether monitoring code quality, deployment success rates, or resource utilization, Grafana is a valuable tool in fostering observability and continuous improvement throughout the software delivery pipeline.

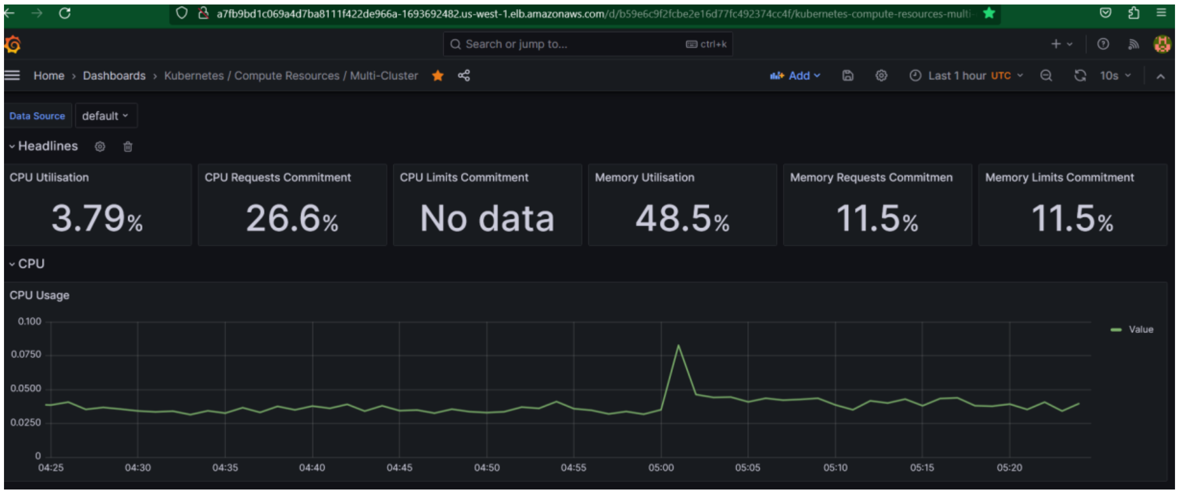

Home › Dashboards › Kubernetes / Compute Resources / Multi-Cluster
Data Source default
Headlines
CPU Utilisation 3.79%
CPU Requests Commitment 26.6%
CPU Limits Commitment No data
Memory Utilisation 48.5%
Memory Requests Commitmen 11.5%
Memory Limits Commitment 11.5%
CPU
CPU Usage
Value

Figure 34 Grafana Dashboard to show Compute Resource

The graphs in Grafana visualize all the information received from Prometheus endpoints, making it easy to monitor resources. We can easily set alerts for changes in resources, which can be used by third-party software or notify DevOps engineers. For instance, we can create an alert when CPU usage exceeds 85% and receive an email or SMS notification when the alert is triggered. This helps us keep a record of changes and monitor resources effectively.

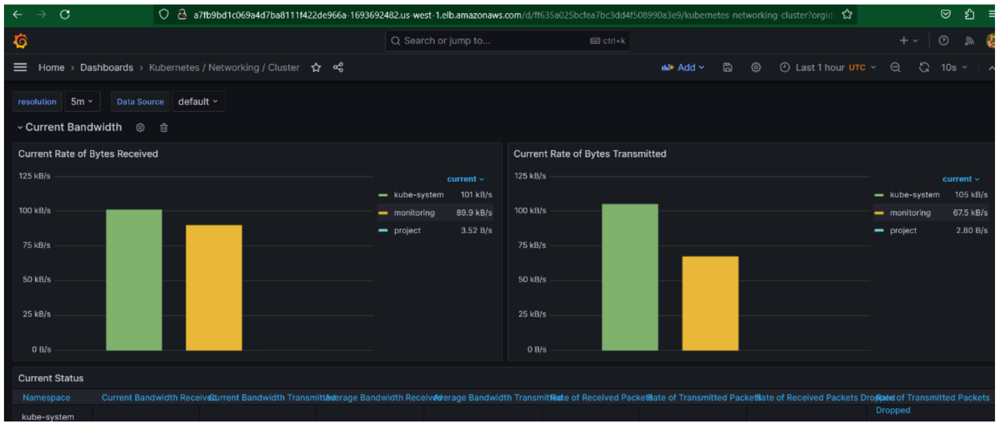


Figure 35 Kubernetes Deployment Bandwidth monitoring

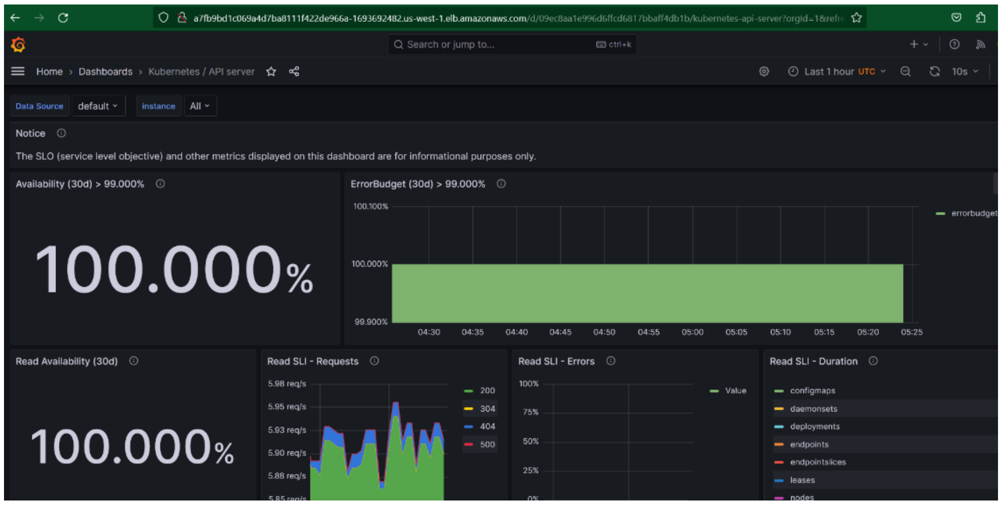


Figure 36 Kubernetes Deployment Availability Chart

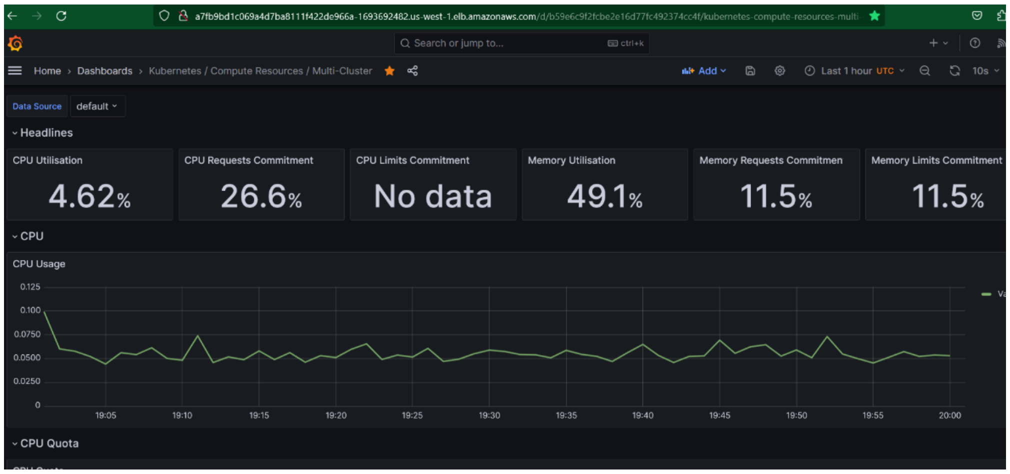

Home › Dashboards › Kubernetes / Compute Resources / Multi-Cluster
Data Source default
Headlines
CPU Utilisation 4.62%
CPU Requests Commitment 26.6%
CPU Limits Commitment No data
Memory Utilisation 49.1%
Memory Requests Commitmen 11.5%
Memory Limits Commitment 11.5%
CPU
CPU Usage
CPU Quota

Figure 37 CPU Utilization for the deployment

5. Conclusion

Moving forward, it is crucial for us to keep improving and expanding upon our innovative approach. The integration of AI into DevOps is a rapidly evolving field, and our journey towards more cost-efficient and error-resistant software development is ongoing. By staying committed to automation, predictive analysis, and continuous improvement, we can effectively meet the ever-changing demands of the software development industry and ensure that our projects remain at the forefront of technological innovation.

5.1 Future Work

Auto-healing in CICD Pipeline: The AI currently has room for improvement, but implementing automated error resolution can help resolve errors with the help of OpenAI. However, any changes made by the AI will require approval from the developer before deployment[44]. Here's how it works: after the code is published on GitHub, the AI will scan it for bugs and attempt to fix them. If a solution is found, the AI will push the corrected code directly onto the branch. It's important to note that there is still a possibility of errors since the AI is still learning to understand and solve complex problems. When it comes to auto healing, we can only automatically heal the deployments because we have control over the Kubernetes manifest files. These files can create a new replica if the older one is not working properly or if there is any kind of error in the replica. For example, if there is an issue with one of the pods or if we accidentally delete a pod, the deployment will recognize that there is only one pod running when there should be two, and it will create a new pod instantly. However, when it comes to complete CI/CD pipeline healing, there is currently no better solution available to completely heal the actual pipeline.

Automated error resolution: Automated error resolution is critical in modern software development. To address this, GitLab has introduced an auto-healing feature in their CI/CD pipeline. This feature leverages the Total Cost:$135.61power of Langchain and OpenAI to automatically detect and fix code errors when a build action fails.

The process is simple:
1. A code push triggers the GitLab CI/CD pipeline.
2. The CI job runs the build script.
3. If the build fails, the self-healing process is triggered.
4. The Langchain and OpenAI algorithms identify, analyze, and fix the error.
5. Changes are automatically committed back to the repository.

Future enhancements to this project may include embedding dependency information, supporting multiple files, implementing retry limits, and creating pull requests for code review.
The goal of this project is to automate error resolution in the CI/CD pipeline, thus saving developers valuable time and effort.

5.2 Limitations

A DevOps project that combines AI for cost optimization and error prediction can be limited by over-reliance on AI models. While AI can significantly improve DevOps practices, it's important to recognize and address the following limitation: The success of cost optimization and error prediction in a DevOps environment depends heavily on the accuracy of AI models. These models are trained on historical data, so their predictions may not always match real-world conditions or emerging trends[45]. Inaccurate predictions could result in suboptimal cost-saving decisions or false alarms regarding potential errors. Therefore, it's crucial to continuously validate and fine-tune AI models to ensure their reliability. Additionally, human oversight and intervention are essential to mitigate the impact of AI inaccuracies, emphasizing the need for a well-balanced human-AI collaboration in the DevOps process[46]. To address this limitation, a robust validation process for AI models, ongoing model training and refinement, and a well-defined protocol for handling situations where AI predictions may not be entirely reliable are necessary.

References

[1] Len Bass, Ingo Weber, and Liming Zhu, “DevOps: A Software Architect’s Perspective,” *IEEE Softw*, vol. 32, no. 6, pp. 40–45, 2015.

[2] Z. Guo, T. Bao, W. Wu, C. Jin, and J. Lee, “IAI DevOps: A Systematic Framework for Prognostic Model Lifecycle Management,” *2019 Prognostics and System Health Management Conference, PHM-Qingdao 2019*, Oct. 2019, doi: 10.1109/PHM-QINGDAO46334.2019.8943069.

[3] Matthias Naab and Alexander Schatten, “A Survey of DevOps Tools,” pp. 429–434, 2016.

[4] S. A. I. B. S. Arachchi and I. Perera, “Continuous integration and continuous delivery pipeline automation for agile software project management,” *MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference*, pp. 156–161, Jul. 2018, doi: 10.1109/MERCON.2018.8421965.

[5] S. A. Ganugapati and S. Prabhu, “Unifying Governance, Risk and Controls Framework Using SDLC, CICD and DevOps,” *Proceedings of the 8th International Conference on Communication and Electronics Systems, ICCES 2023*, pp. 1797–1802, 2023, doi: 10.1109/ICCES57224.2023.10192730.

[6] Y. Bhalla, V. Hemamalini, and S. Mishra, “Automating Hadoop Cluster on Aws Cloud Using Terraform,” *Proceedings of the 1st IEEE International Conference on Networking and Communications 2023, ICNWC 2023*, 2023, doi: 10.1109/ICNWC57852.2023.10127568.

[7] “Mastering Ansible: Automate configuration management and overcome deployment challenges with Ansible | Packt Publishing books | IEEE Xplore.” Accessed: Oct. 19, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10162445

[8] S. Mysari and V. Bejgam, “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible,” *International Conference on Emerging Trends in Information Technology and Engineering, ic-ETITE 2020*, Feb. 2020, doi: 10.1109/IC-ETITE47903.2020.239.

[9] H. Zhao, H. Lim, M. Hanif, and C. Lee, “Predictive Container Auto-Scaling for Cloud-Native Applications,” *ICTC 2019 - 10th International Conference on ICT Convergence: ICT Convergence Leading the Autonomous Future*, pp. 1280–1282, Oct. 2019, doi: 10.1109/ICTC46691.2019.8939932.

[10] D. Sitaram *et al.*, “Orchestration Based Hybrid or Multi Clouds and Interoperability Standardization,” *Proceedings - 7th IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2018*, pp. 67–71, Jul. 2019, doi: 10.1109/CCEM.2018.00018.

[11] A. Dewey and A. Block, “Managing Kubernetes resources using Helm : simplifying how to build, package and distribute applications for Kubernetes”.

[12] C. Gan, N. Wang, Y. Yang, D. Y. Yeung, and A. G. Hauptmann, “DevNet: A Deep Event Network for multimedia event detection and evidence recounting,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June-2015, pp. 2568–2577, Oct. 2015, doi: 10.1109/CVPR.2015.7298872.

[13] N. Petrovic, “Chat GPT-Based Design-Time DevSecOps,” *2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST 2023 - Proceedings*, pp. 143–146, 2023, doi: 10.1109/ICEST58410.2023.10187247.

[14] ALY. SALEH, “KUBERNETES IN PRODUCTION BEST PRACTICES : build and manage highly available production-ready ... kubernetes clusters.,” 2021.

[15] J. García, J. J. Minero, and E. Lara, “Implementation of Azure DevOps to automate the processes of the ISO/IEC 29110 standard a Case Study,” *Applications in Software Engineering - Proceedings of the*

11th International Conference on Software Process Improvement, CIMPS 2022, pp. 29–36, 2022, doi: 10.1109/CIMPS57786.2022.10035670.

[16] A. Bijwe and P. Shankar, "Challenges of Adopting DevOps Culture on the Internet of Things Applications - A Solution Model," *Proceedings of International Conference on Technological Advancements in Computational Sciences, ICTACS 2022*, pp. 638–645, 2022, doi: 10.1109/ICTACS56270.2022.9988182.

[17] N. Kebbani, P. Tylenda, and R. McKendrick, "The Kubernetes Bible: The definitive guide to deploying and managing Kubernetes across major cloud platforms," 2022, Accessed: Oct. 19, 2023. [Online]. Available: https://books.google.co.id/books?id=i3xZEAAAQBAJ

[18] G. Agarwal, "Modern devops tips, tricks, and techniques manage, secure, and enhance software development in the public cloud with cutting-edge tools and solutions".

[19] D. B. Bose, A. Rahman, and S. I. Shamim, "'Under-reported' Security Defects in Kubernetes Manifests," *Proceedings - 2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems, EnCyCriS 2021*, pp. 9–12, Jun. 2021, doi: 10.1109/ENCYCRIS52570.2021.00009.

[20] "Software and Systems Engineering Standards: Implementing DevOps Best Practices | IEEE Courses | IEEE Xplore." Accessed: Dec. 05, 2023. [Online]. Available: https://ieeexplore.ieee.org/courses/details/EDP655

[21] M. Kersten, "A cambrian explosion of DevOps tools," *IEEE Softw*, vol. 35, no. 2, pp. 14–17, Mar. 2018, doi: 10.1109/MS.2018.1661330.

[22] Sandeep. Madamanchi, "Google Cloud for DevOps Engineers : A Practical Guide to SRE and Achieving Google's Professional Cloud DevOps Engineer Certification.," 2021.

[23] K. German and O. Ponomareva, "An Overview of Container Security in a Kubernetes Cluster," pp. 283–285, Jun. 2023, doi: 10.1109/USBEREIT58508.2023.10158865.

[24] L. E. Lwakatare, I. Crnkovic, and J. Bosch, "DevOps for AI - Challenges in Development of AI-enabled Applications," *2020 28th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2020*, Sep. 2020, doi: 10.23919/SOFTCOM50211.2020.9238323.

[25] S. M. A. G. Paul M. Duvall, "Continuous Integration: Improving Software Quality and Reducing Risk," *Addison-Wesley*, 2007.

[26] G. Sayfan, "Mastering Kubernetes : dive into Kubernetes and learn how to create and operate world-class cloud-native systems," p. 744.

[27] ALY. SALEH, "KUBERNETES IN PRODUCTION BEST PRACTICES : build and manage highly available production-ready ... kubernetes clusters.," 2021.

[28] Subhajit. Chatterjee, Swapneel. Deshpande, Henry. Been, and M. van der. Gaag, "Designing and Implementing Microsoft DevOps Solutions AZ-400 Exam Guide : Prepare for the certification exam and successfully apply Azure DevOps strategies with practical labs".

[29] J. Knight and N. Swenson, "The DevOps Career Handbook The Ultimate Guide to Pursuing a Successful Career in DevOps.".

[30] Gineesh. Madapparambath, "Ansible for Real-Life Automation : A complete Ansible handbook filled with practical IT automation use cases".

[31] L. Chen, M. Xian, and J. Liu, "Monitoring System of OpenStack Cloud Platform Based on Prometheus," *Proceedings - 2020 International Conference on Computer Vision, Image and Deep Learning, CVIDL 2020*, pp. 206–209, Jul. 2020, doi: 10.1109/CVIDL51233.2020.0-100.

[32] Mikael. Krief, "Learning DevOps : A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins".

[33] James. Freeman and Jesse. Keating, "Mastering Ansible : automate configuration management and overcome deployment challenges with Ansible".

[34] "Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform | Manning books | IEEE Xplore." Accessed: Dec. 04, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10280525

[35] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2018-July, pp. 970–973, Sep. 2018, doi: 10.1109/CLOUD.2018.00148.

[36] G. Rostami, "Role-based Access Control (RBAC) Authorization in Kubernetes," *Journal of ICT Standardization*, Sep. 2023, doi: 10.13052/JICTS2245-800X.1132.

[37] Alexander. Raul, "Cloud native with bubernetes : deploy, configure, and run modern cloud native applications on Kubernetes," p. 446.

[38] K. Paul, "Design features of the helms pumped storage project," *IEEE Transactions on Energy Conversion*, vol. 4, no. 1, pp. 9–15, 1989, doi: 10.1109/60.23143.

[39] K. Gajananan, H. Kitahara, R. Kudo, and Y. Watanabe, "Scalable Runtime Integrity Protection for Helm Based Applications on Kubernetes Cluster," *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, pp. 2362–2371, 2021, doi: 10.1109/BIGDATA52589.2021.9671944.

[40] A. Zerouali, R. Opdebeeck, and C. De Roover, "Helm Charts for Kubernetes Applications: Evolution, Outdatedness and Security Risks," *Proceedings - 2023 IEEE/ACM 20th International Conference on Mining Software Repositories, MSR 2023*, pp. 523–533, 2023, doi: 10.1109/MSR59073.2023.00078.

[41] Y. Guo and W. Yao, "A container scheduling strategy based on neighborhood division in micro service," *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, pp. 1–6, Jul. 2018, doi: 10.1109/NOMS.2018.8406285.

[42] R. Mishra and an O. M. Company. Safari, "HashiCorp Infrastructure Automation Certification Guide," p. 350, 2021.

[43] S. Gokhale *et al.*, "Creating Helm Charts to ease deployment of Enterprise Application and its related Services in Kubernetes," *2021 International Conference on Computing, Communication and Green Engineering, CCGE 2021*, 2021, doi: 10.1109/CCGE50943.2021.9776450.

[44] S. A. I. B. S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," *MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference*, pp. 156–161, Jul. 2018, doi: 10.1109/MERCON.2018.8421965.

[45] G. Sayfan, "Mastering Kubernetes : dive into Kubernetes and learn how to create and operate world-class cloud-native systems," p. 744.

[46] R. Eramo *et al.*, "AIdoArt: AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems," *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*, pp. 303–310, 2021, doi: 10.1109/DSD53832.2021.00053.