

# **DOCKIFY**

(Deploying an application on Docker and Kubernetes with Jenkins CI/CD)

**Enrollment Numbers:** 21103105, 21103102, 21103096

**Name of Students:** Mukund Sarda, Priyanshu Jain, Karanjot Singh

**Name of Supervisor:** Dr. Amarjeet Prajapati



**December – 2023**

**Submitted in partial fulfilment of the degree of**

**Bachelor of Technology**

**In**

**Computer Science Engineering**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND  
INFORMATION TECHNOLOGY**

**JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY**

(I)

**TABLE OF CONTENTS**

<b>Part No.</b>	<b>Description</b>	<b>Page No.</b>
<b>II</b>	Declaration	4
<b>III</b>	Certificate	5
<b>IV</b>	Acknowledgement	6
<b>V</b>	Summary	7
<b>VI</b>	List of Figures	8
<b>VII</b>	List of Symbols and Acronyms	9
<b>VIII</b>	List of Tables	10
<b>Chapter No.</b>	<b>Topics</b>	<b>Page No.</b>
<b>Chapter-1</b>	<b>Introduction</b>	11-18
	1.1 General Introduction	11-12
	1.2 Problem Statement	13
	1.3 Significance of the Problem	14
	1.4 Empirical Study	15
	1.5 Brief Description of Solution Approach	16
	1.6 Comparison of Existing Approaches of the Problem Framed	17-18
<b>Chapter-2</b>	<b>Literature Summary</b>	19-22
	2.1 Summary of Papers Studied	19-21
	2.2 Integrated Summary of the Literature Studied	22
<b>Chapter-3</b>	<b>Requirement Analysis and Solution Approach</b>	23-30
	3.1 Overall Description of the Project	23

	3.2 Requirement Analysis	24-25
	3.3 Solution Approach	26-30
<b>Chapter-4</b>	<b>Modelling and Implementation Details</b>	31-35
	4.1 Design Diagrams	31-33
	4.2 Implementation Details and Issues	34-36
<b>Chapter-5</b>	<b>Testing</b>	37-41
	5.1 Testing Plan	37
	5.2 Components Decomposition and Type of Testing Required	38
	5.3 List of Tests	39-40
	5.4 Limitations of the Solution	41
<b>Chapter-6</b>	<b>Findings, Conclusion and Future Work</b>	42-46
	6.1 Findings	42-43
	6.2 Conclusion	44
	6.3 Future work	45-46
	<b>REFERENCES</b>	47

(II)

**DECLARATION**

We, Mukund Sarda, Priyanshu Jain, and Karanjot Singh, hereby declare that this submission is our own work and that, to the best of my knowledge and belief, it contains no material previously or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

-----  
Mukund Sarda

(21103105)

-----  
Priyanshu Jain

(21103102)

-----  
Karanjot Singh

(21103096)

**Place:** IIIT, Noida, India

**Date:** 2 December, 2023

(III)

**CERTIFICATE**

This is to certify that the work titled “**DOCIFY- Deploying an application on Docker and Kubernetes with Jenkins CI/CD**” submitted by **Mukund Sarda, Priyanshu Jain, and Karanjot Singh**, in partial fulfillment for the award of degree of B. Tech in Computer Science of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to another University or Institute for the award of this or other degree or diploma.

-----

Dr. Amarjeet Prajapati

(Assistant Professor- Senior Grade)

**Date:** 2 December, 2023

(IV)

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude and heartfelt thanks to everyone who has contributed to the successful completion of this minor project.

First and foremost, we extend our deepest appreciation to, Dr Amarjeet Prajapati, our project supervisor, for his invaluable guidance, unwavering support, and insightful feedback throughout the duration of this project. Their expertise and encouragement have been instrumental in shaping the project and enhancing its quality.

We would also like to thank, Jaypee Institute of Information Technology, for providing the necessary resources and conducive environment for conducting this project. The library and laboratory facilities have played a crucial role in the research and development process.

We would like to thank family and friends for their continuous encouragement and understanding during the challenging phases of this project. Their belief in our abilities has been a driving force behind our perseverance.

This project has been a significant learning experience, and we are truly grateful to have had the opportunity to work on it. Thank you all for being an integral part of this journey.

-----  
Mukund Sarda

(21103105)

-----  
Priyanshu Jain

(21103102)

-----  
Karanjot Singh

(21103096)

**Date:** 2 December, 2023

(V)

## SUMMARY

Our project 'DOCKIFY' seamlessly deploys the application, utilizing Jenkins as a CI/CD tool for automation, Docker for efficient containerization, and Kubernetes for scalable orchestration. Initiated by launching an Ubuntu instance on AWS, the project incorporates Jenkins pipelines to automate build, testing, and deployment phases. Docker containers ensure portability and consistency, while Kubernetes orchestrates containerized applications for scalability. The integration of security measures, including Trivy vulnerability scanning, enhances the overall robustness of the deployment. Through this project, the team gains proficiency in Linux and Docker commands, as well as hands-on experience with JavaScript and React for development. The documentation phase, encapsulates insights, ensuring knowledge transfer and facilitating future improvements. Overall, this project highlights the synergy of modern DevOps practices and containerization technologies in achieving a streamlined and automated deployment of the application.

-----  
Mukund Sarda

(21103105)

-----  
Priyanshu Jain

(21103102)

-----  
Karanjot Singh

(21103096)

-----  
Dr. Amarjeet Prajapati

**Date:** 2 December, 2023

(VI)

**LIST OF FIGURES**

<b>Figure</b>	<b>Description</b>	<b>Page No.</b>
Figure 1.1	Docker Architecture and Commands	12
Figure 3.1	Running 2048 Web Application Using Docker	26
Figure 3.2	Setting Up Aws Ec2 Instance	26
Figure 3.3	Initialising Jenkins Pipeline	27
Figure 3.4	Running Different Containers Inside Docker	27
Figure 3.5	Configuring Trivy for Vulnerability Scanning	28
Figure 3.6	Installing And Testing Sonarqube and Quality Gate in Jenkins	28
Figure 3.7	Sonarqube Analysis Result	29
Figure 3.8	OWASP Dependency Check Console Output	29
Figure 3.9	Jenkins Successful Pipeline Build	30
Figure 3.10	2048 Game Successful Build and Run	30
Figure 4.1	Use Case Diagram	31
Figure 4.2	Application Design Diagram	32
Figure 4.3	Kubernetes Deployment Diagram	32
Figure 4.4	Trivy Security Scanning Outline	33
Figure 4.5	Activity Diagram	33
Figure 5.1	Sending API Requests Using Locust	39
Figure 5.2	Configuring Locust	39
Figure 5.3	Requests Per Second Summary	40
Figure 5.4	Response Time Summary	40
Figure 5.5	Application Response Time Graph W.R.T Other API Requests	40



(VII)

**LIST OF SYMBOLS AND ACRONYMS**

Symbols	Meaning
k8s	Kubernetes
CI/CD	Continuous Integration and Development
jdk	Java Development Kit
Ops	Operation
dev	Development
Aws ec2	Amazon Elastic Compute Cloud

(VIII)

**LIST OF TABLES**

<b>Table</b>	<b>Description</b>	<b>Page No.</b>
Table 4.1	Commands Used	36

## CHAPTER- 1

### INTRODUCTION

#### **1.1) GENERAL INTRODUCTION**

**DevOps:** DevOps is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development (Dev) and IT operations (Ops) as a means for improving and shortening the systems development life cycle.

Proposals to combine software development methodologies with deployment and operations concepts began to appear in the late 80s and early 90s.

Around 2007 and 2008, concerns were raised by those within the software development and IT communities that the separation between the two industries, where one wrote and created software entirely separate from those that deploy and support the software, was creating a fatal level of dysfunction within the industry.

In 2009, the first conference named DevOps Days was held in Ghent, Belgium. The conference was founded by Belgian consultant, project manager and agile practitioner Patrick Debois. The conference has now spread to other countries.

In 2012, a report called “State of DevOps” was first published by Alanna Brown at Puppet Labs.

As of 2014, the annual State of DevOps report was published by Nicole Forsgren, Gene Kim, Jez Humble and others. They stated that the adoption of DevOps was accelerating. Also in 2014, Lisa Crispin and Janet Gregory wrote the book More Agile Testing, containing a chapter on testing and DevOps.

**Docker:** Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security let you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't

need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data centre, a cloud provider, or a hybrid of the two.

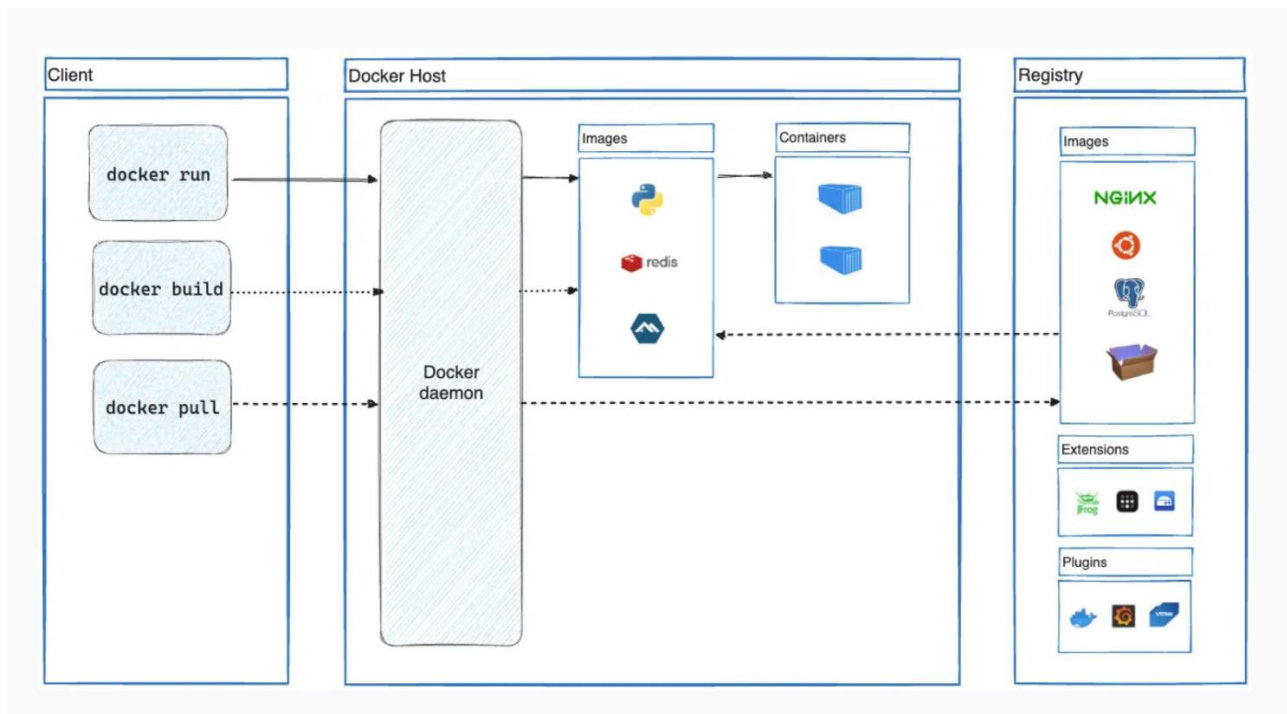


Fig 1.1

## 1.2) PROBLEM STATEMENT

In traditional deployment methods, organizations encounter multifaceted challenges that have underscored the imperative need for embracing modern DevOps practices. These challenges significantly impede the efficiency, scalability, and reliability of software development and deployment processes.

**1. Inconsistency Across Environments:** Traditional deployment struggles to ensure uniformity between development, testing, and production environments, leading to compatibility issues and discrepancies that impede a smooth deployment process.

**2. Manual and Error-Prone Processes:** Manual intervention in deployment processes not only consumes valuable time but also introduces a heightened risk of errors. These errors can result in deployment failures, system downtimes, and compromised software quality.

**3. Scalability Concerns:** Traditional methods often lack the automation necessary for seamless scaling of applications in response to varying workloads. Scaling becomes a manual and resource-intensive process, hindering adaptability to changing demands.

**4. Resource Inefficiencies:** Inefficient resource utilization is a persistent challenge, as traditional deployment models may not optimize infrastructure resources effectively. This leads to increased operational costs and underutilized computing capacity.

**5. Security Vulnerabilities:** Inconsistent environments and manual configurations in traditional deployment methods expose applications to security vulnerabilities. Ensuring the security of applications throughout the deployment lifecycle becomes a complex and challenging endeavour.

### **1.3) SIGNIFICANCE OF THE PROBLEM**

In the ever-evolving landscape of software development and deployment, traditional methods often face challenges in terms of scalability, consistency, and efficiency. The advent of technologies like Docker, Kubernetes, and modern CI/CD tools like Jenkins has revolutionized the software development lifecycle, offering compelling reasons to transition from old methods to these contemporary approaches.

#### **1. Streamlined Deployment:**

- Docker simplifies application packaging and ensures consistent runtime across diverse environments, eliminating the infamous "it works on my machine" issue.
- Kubernetes orchestrates containerized applications, providing automated scaling, load balancing, and simplified deployment management, streamlining the entire process.

#### **2. Scalability and Resource Optimization:**

- Containerization with Docker allows for easy scaling by replicating containers, making it straightforward to handle varying workloads.
- Kubernetes provides an automated and efficient way to scale applications based on demand, optimizing resource utilization and reducing infrastructure costs.

#### **3. Continuous Integration and Deployment:**

- Jenkins automates the CI/CD pipeline, enabling continuous integration, testing, and delivery, fostering faster and more reliable software releases.

#### **4. Portability and Consistency:**

- Docker containers encapsulate applications and their dependencies, ensuring consistent behaviour across development, testing, and production environments.
- Kubernetes abstracts the underlying infrastructure, allowing for seamless portability of applications across different hosting environments.

#### **5. Enhanced Security:**

- Docker's containerization enhances security by isolating applications and dependencies, reducing the attack surface.
- Kubernetes incorporates features for automated updates, rollbacks, and resource isolation, contributing to a more secure deployment environment.

## 1.4) EMPIRICAL STUDY

### Real-Life Analysis: How DevOps Transformed Industries

**1. Faster Time-to-Market at Amazon:** Amazon, a pioneer in e-commerce, embraced DevOps to enhance its time-to-market for new features and services. By implementing continuous integration and delivery pipelines, Amazon shortened release cycles, enabling quicker responses to market demands. This increased agility has been a key factor in maintaining Amazon's competitive edge.

**2. Continuous Innovation at Netflix:** Netflix, a global streaming giant, credits DevOps for its ability to continuously innovate and release new content features. DevOps practices at Netflix facilitate automated testing, seamless deployment, and quick rollbacks, ensuring a consistently smooth streaming experience for millions of users worldwide.

**3. Reliability and Uptime at Etsy:** Etsy, an e-commerce platform, turned to DevOps to improve system reliability and minimize downtime. Through infrastructure as code (IaC) and automated deployment processes, Etsy achieved greater stability in its systems, reducing the impact of potential issues on the platform's performance.

**4. Improved Collaboration at Target:** Retail giant Target adopted DevOps to foster collaboration between its development and operations teams. By breaking down silos and implementing shared practices, Target achieved faster releases, improved communication, and more effective issue resolution. This collaborative approach contributed to enhanced overall operational efficiency.

**5. Efficiency Gains at Google:** Google, a technology behemoth, integrated DevOps practices to streamline its software development lifecycle. Continuous integration and automated testing have allowed Google to identify and fix issues early in the development process, leading to more efficient and reliable software releases across various Google services.

## **1.5) BRIEF DESCRIPTION OF SOLUTION APPROACH**

The solution approach revolves around modernizing the software development and deployment processes to overcome challenges inherent in traditional methods. Embracing a holistic and collaborative methodology, the project focuses on enhancing efficiency, reliability, and adaptability.

### **1. Streamlined Automation:**

- The approach emphasizes the implementation of streamlined automation, reducing manual interventions and minimizing the scope for errors. This ensures a more efficient and error-free software delivery process.

### **2. Consistency and Portability:**

- Containerization is employed to encapsulate the application and its dependencies, fostering consistency and portability across different environments. This approach aims to eliminate discrepancies often encountered in traditional deployment methods.

### **3. Efficient Orchestration:**

- Orchestrating the deployment process ensures efficient scaling and resource optimization without delving into specific technical intricacies. The focus is on achieving a smoother and more adaptable deployment workflow.

### **4. Security Enhancement:**

- Security measures are integrated to fortify the overall security posture of the deployment. The approach aims to enhance security without detailing the technical nuances, ensuring a resilient and protected software environment.

In essence, the solution approach prioritizes a strategic shift towards modern DevOps practices, fostering collaboration and automation without delving into the intricate technicalities. This approach aims to provide a more efficient, reliable, and adaptable framework for software development and deployment.



## **1.6) COMPARISON OF EXISTING APPROACHES TO THE PROBLEM FRAMED**

Our Modern DevOps Solution vs. Traditional Methods

### **1. Automation:**

- Traditional Methods: Manual deployment processes characterized by human intervention, leading to time-consuming tasks and increased likelihood of errors.
- Our Solution: Emphasizes automation through CI/CD pipelines, reducing manual interventions, and ensuring a consistent and error-free deployment.

### **2. Consistency and Portability:**

- Traditional Methods: Inconsistencies in runtime environments across development, testing, and production stages, leading to compatibility issues.
- Our Solution: Utilizes containerization (e.g., Docker) to encapsulate applications and dependencies, ensuring consistency and portability, reducing discrepancies.

### **3. Scalability:**

- Traditional Methods: Manual scaling processes, often resource-intensive and prone to errors, limiting adaptability to varying workloads.
- Our Solution: Embraces orchestration tools (e.g., Kubernetes) for automated scaling, providing efficiency and adaptability to changing demands seamlessly.

### **4. Resource Utilization:**

- Traditional Methods: Inefficient resource utilization due to manual configuration and deployment processes.
- Our Solution: Optimizes resource utilization through automation and containerization, minimizing operational costs and maximizing efficiency.

### **5. Security:**

- Traditional Methods: Manual security configurations, potential vulnerabilities due to inconsistent environments.
- Our Solution: Integrates security measures (e.g., vulnerability scanning) into the CI/CD pipeline, enhancing overall security posture without manual intervention.

## **6. Collaboration:**

- Traditional Methods: Siloed development and operations teams, leading to communication gaps and delays in issue resolution.

- Our Solution: Fosters collaboration through shared practices, breaking down silos, and promoting cross-functional teams for more effective communication and issue resolution.

## **7. Time-to-Market:**

- Traditional Methods: Longer release cycles, delayed responses to market demands.

- Our Solution: Shortened release cycles through continuous integration and delivery, enabling quicker responses to changing market dynamics.

## **8. Adaptability:**

- Traditional Methods: Rigidity in adapting to changes and accommodating new features.

- Our Solution: Greater adaptability to changes and continuous innovation, facilitated by automated processes and efficient scaling mechanisms.

## **9. Documentation and Knowledge Transfer:**

- Traditional Methods: Limited documentation, resulting in knowledge gaps and difficulties in knowledge transfer.

- Our Solution: Prioritizes comprehensive documentation, capturing insights, lessons learned, and facilitating knowledge transfer for continuous improvement.

In summary, our Modern DevOps Solution represents a paradigm shift from manual and traditional methods towards automation, consistency, scalability, and enhanced collaboration. These advancements contribute to increased efficiency, reliability, and agility in software development and deployment processes.

## **CHAPTER- 2**

### **LITERATURE SURVEY**

#### **2.1) SUMMARY OF PAPERS STUDIED**

##### **1. An Introduction to Docker and Analysis of its Performance**

**- Bashari Rad, Babak & Bhatti, Harrison & Ahmadi, Mohammad (2017)**

Docker provides some facilities, which are useful for developers and administrators. It is an open platform that can be used for building, distributing, and running applications in a portable, lightweight runtime and packaging tool, known as Docker Engine. It also provides Docker Hub, which is a cloud service for sharing applications. Costs can be reduced by replacing traditional virtual machines with docker containers. It excellently reduces the cost of rebuilding the cloud development platform.

##### **2. Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research**

**- Ferme, Vincenzo - Gall, Harald**

The ability to replicate and reproduce scientific results has become an increasingly important topic for many academic disciplines. In computer science and, more specifically, software and web engineering, contributions of scientific work rely on developed algorithms, tools and prototypes, quantitative evaluations, and other computational analyses. Published code and data come with many undocumented assumptions, dependencies, and configurations that are internal knowledge and make reproducibility hard to achieve. This report presents how Docker containers can overcome these issues and aid the reproducibility of research artifacts in software and web engineering and discusses their applications in the field.

##### **3. DevOps - Software Development Methodology**

**~Velibor Božić**

DevOps is a software development methodology that focuses on collaboration, communication, and integration between software development teams and IT operations teams. The goal of DevOps

is to streamline the software development process, reduce time-to-market, and increase the reliability and quality of software releases.

#### **4. A Qualitative Study of DevOps Usage in Practice. Journal of Software: Evolution and Process**

**- Erich, Floris & Amrit, Chintan & Daneva, Maya (2017)**

Organizations are introducing agile and lean software development techniques in operations to increase the pace of their software development process and to improve the quality of their software. They use the term DevOps, a portmanteau of development and operations, as an umbrella term to describe their efforts. In this paper we describe the ways in which organizations implement DevOps and the outcomes they experience. We first summarize the results of a Systematic Literature Review that we performed to discover what researchers have written about DevOps. We then describe the results of an exploratory interview-based study involving six organizations of various sizes that are active in various industries. As part of our findings, we observed that all organizations were positive about their experiences and only minor problems were encountered while adopting DevOps.

#### **5. Modelling Performance & Resource Management in Kubernetes**

**- Medel, Víctor & Rana, Omer & Bañares, José & Arronategui, Unai (2016)**

Containers are rapidly replacing Virtual Machines (VMs) as the compute instance of choice in cloud-based deployments. The significantly lower overhead of deploying containers (compared to VMs) has often been cited as one reason for this. We analyse performance of the Kubernetes system and develop a Reference net-based model of resource management within this system. Our model is characterized using real data from a Kubernetes deployment, and can be used as a basis to design scalable applications that make use of Kubernetes.

#### **6. Jenkins-CI, an Open-Source Continuous Integration System**

**- Moutsatsos, Ioannis & Hossain, Imtiaz & Agarinis, (2016)**

Large volumes of diverse data generated by high-throughput screening. The study emphasized the challenges of building integrated and scalable workflows for scientific data and image processing,

highlighting the value of standardized processing, collaboration, and the reuse of best practices. Jenkins-CI was employed to construct pipelines for processing images and associated data from high-content screening (HCS). Leveraging Jenkins-CI's plugins for standard compute tasks and its ability to integrate external scientific applications, the researchers successfully integrated CellProfiler, an open-source image-processing platform, with various HCS utilities and a high-performance Linux cluster. The platform, accessible through the web, facilitated efficient access and sharing of high-performance computer resources, automating previously cumbersome data and image-processing tasks. The study also addressed limitations in Jenkins-CI, particularly related to the user interface, by selecting helper plugins from the Jenkins-CI community. The resulting platform enabled the management, sharing, and annotation of imaging pipelines developed using the CellProfiler client through a centralized Jenkins-CI repository, fostering collaboration and reuse of pipelines and data.

## **7. DevOps for Manufacturing Systems: Speeding Up Software Development**

**- Blüher, Till & Maelzer, Daniel & Harrendorf, Jessica & Stark, Rainer (2023)**

The growing significance of software as a fundamental provider in various products and processes necessitates companies to excel in development capabilities, ensuring the continuous and swift delivery of high-quality software features. DevOps, derived from Development and Operations, emerges as a crucial approach in mastering these capabilities, predominantly utilized in software-driven industries thus far. This study delves into the exploration of conditions that facilitate the extension of DevOps into the industrial context of series manufacturing. The goal is to enable the consistent development and delivery of software features tailored for industrial environments, such as monitoring and maintaining manufacturing machines. Building upon current best practices and the unique dynamics of industrial settings, a DevOps concept for manufacturing systems was formulated. This concept was subsequently put into action and validated with experts through initial development cycles, demonstrating the efficacy and applicability of DevOps for enhancing manufacturing systems.

## **2.2) INTEGRATED SUMMARY OF LITERATURE STUDIED**

The texts cover diverse aspects of software development and technological applications. Docker is introduced as an open platform, offering cost-effective alternatives for building and running applications. Its advantages, particularly in cloud development, are highlighted. Another focus is on Docker's role in improving reproducibility in scientific research, addressing challenges in replicating and reproducing results in software and web engineering.

DevOps emerges as a pivotal methodology, emphasizing collaboration, communication, and integration between development and IT operations teams. Its overarching goal is to streamline software development, reduce time-to-market, and enhance software release reliability and quality. The study explores how organizations implement DevOps, showcasing positive experiences with minor challenges.

The analysis extends to Kubernetes, where containers are rapidly replacing Virtual Machines. The study models Kubernetes' performance and resource management, providing a reference net-based model derived from real data. This model serves as a basis for designing scalable applications leveraging Kubernetes.

Jenkins-CI is examined as an open-source continuous integration system for scientific data and image processing. The study underscores standardized processing, collaboration, and the reuse of best practices. Jenkins-CI is utilized to construct pipelines, integrating external scientific applications and addressing limitations through community plugins.

The final text delves into DevOps in the context of manufacturing systems. It explores the application of DevOps in series manufacturing, formulating a concept based on best practices. The concept is implemented, validated with experts, and demonstrates efficacy in accelerating software development for monitoring and maintaining manufacturing machines.

## **CHAPTER- 3**

### **REQUIREMENT ANALYSIS AND SOLUTION APPROACH**

#### **3.1) OVERALL DESCRIPTION OF THE PROJECT**

1. Commence by launching a T2 Large Instance with Ubuntu 22.04, establishing the groundwork for subsequent steps in the deployment process.
2. Install crucial components—Jenkins, Docker, and Trivy—and create a Dockerized Sonarqube Container, fortifying the development environment and ensuring a seamless testing framework.
3. Enhance functionality by installing essential plugins, including JDK, Sonarqube Scanner, Nodejs, and OWASP Dependency Check, bolstering the development pipeline with diverse capabilities.
4. Foster automation prowess by crafting a Jenkins Pipeline Project through a Declarative Pipeline, streamlining the development workflow and ensuring efficient code integration.
5. Augment security measures by incorporating OWASP Dependency Check Plugins, fortifying the software against potential vulnerabilities and ensuring a robust codebase.
6. Execute the building and pushing of Docker images, optimizing the containerization process for enhanced efficiency in deployment.
7. Seamlessly deploy the Docker image, ensuring a smooth transition from development to deployment, and facilitating the application's availability.
8. Propel scalability and management efficiency by establishing Kubernetes master and slave nodes on Ubuntu 20.04, paving the way for orchestration and streamlined container management.
9. Extend accessibility by enabling users to access the game through a web browser, enhancing user experience and interaction.
10. Conclude the deployment cycle by terminating AWS EC2 Instances, optimizing resource utilization and ensuring cost-effective cloud management.

## **3.2) REQUIREMENT ANALYSIS**

### **1. Infrastructure Requirements:**

- AWS EC2 Instances: Provision T2 Large Instances with Ubuntu 22.04 for the deployment environment.
- Kubernetes Cluster: Set up master and slave nodes on Ubuntu 20.04 for efficient container orchestration.
- Networking: Ensure proper network configurations to facilitate communication between instances and enable external access.

### **2. Software and Tools:**

- Jenkins: Install Jenkins for continuous integration and deployment, creating a seamless CI/CD pipeline.
- Docker: Implement Docker for containerization, allowing the packaging and distribution of the 2048 Game application and its dependencies.
- Trivy: Install Trivy for vulnerability scanning of Docker images, ensuring security best practices.
- Sonarqube: Utilize Sonarqube for continuous code quality checks and analysis.
- OWASP Dependency Check: Integrate OWASP Dependency Check Plugins to identify and address open-source component vulnerabilities.
- Kubernetes: Set up Kubernetes for container orchestration, scalability and efficient management.

### **3. Development and Deployment Dependencies:**

- JDK: Install Java Development Kit to support Java-based development components.
- Nodejs: Incorporate Nodejs for JavaScript runtime, enabling the execution of JavaScript code during the deployment process.

### **4. Jenkins Pipeline Configuration:**

- Declarative Pipeline: Implement a Declarative Pipeline in Jenkins for a structured and streamlined deployment workflow.
- Plugins: Ensure the installation of Jenkins plugins for Sonarqube integration, Docker image building, and OWASP Dependency Check.



## **5. Game Access Requirements:**

- Web Browser: Guarantee compatibility with common web browsers for users to access and interact with the deployed 2048 Game.

## **6. Documentation:**

- Centralized Repository: Establish a centralized repository for managing documentation, including insights, lessons learned, and recommendations.
- Knowledge Transfer: Facilitate knowledge transfer by documenting the deployment process, configurations, and best practices for future reference.

## **7. Security Measures:**

- User Authentication: Implement user authentication mechanisms to secure access to Jenkins and other sensitive components.
- Firewall Configuration: Configure firewalls on AWS instances to control incoming and outgoing traffic, enhancing overall system security.

## **8. Scalability and Resource Optimization:**

- Kubernetes Autoscaling: Configure Kubernetes to enable autoscaling, ensuring efficient resource utilization based on demand.

## **9. Termination Process:**

- Graceful Shutdown: Implement a graceful termination process for AWS EC2 Instances, minimizing potential disruptions during termination.

## **10. User Acceptance Testing (UAT):**

- Test Environment: Set up a dedicated test environment to conduct UAT before deploying to the production environment.
- Test Cases: Develop comprehensive test cases to validate the functionality, performance, and security aspects of the deployed 2048 Game.

This comprehensive requirement analysis outlines the essential components, configurations, and considerations for successfully deploying the application on Docker and Kubernetes with Jenkins CI/CD.

### 3.3) SOLUTION APPROACH

#### 1. Running the application

```
karanjotsingh@itsfuckindorean>
docker compose up
Attaching to 2048-react-cicd-node-app-1, 2048-react-cicd-webserver-1
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
2048-react-cicd-webserver-1 | 10-listen-on-ipv6-by-default.sh: info: IPv6 listen already enabled
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
2048-react-cicd-webserver-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: using the "epoll" event method
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: nginx/1.25.3
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: OS: Linux 5.15.49-linuxkit
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: start worker processes
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: start worker process 22
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: start worker process 23
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: start worker process 24
2048-react-cicd-webserver-1 | 2023/11/30 18:39:21 [notice] 1#1: start worker process 25
2048-react-cicd-node-app-1 |
2048-react-cicd-node-app-1 | > 2048-in-react@0.1.0 start
2048-react-cicd-node-app-1 | > craco start
2048-react-cicd-node-app-1 |
2048-react-cicd-node-app-1 | i [wds]: Project is running at http://172.19.0.3/
2048-react-cicd-node-app-1 | i [wds]: webpack output is served from /2048-in-react
2048-react-cicd-node-app-1 | i [wds]: Content not from webpack is served from /app/public
2048-react-cicd-node-app-1 | i [wds]: 404s will fallback to /2048-in-react/
2048-react-cicd-node-app-1 | Starting the development server...
2048-react-cicd-node-app-1 |
2048-react-cicd-node-app-1 | Compiled successfully!
2048-react-cicd-node-app-1 | You can now view 2048-in-react in the browser.
2048-react-cicd-node-app-1 |
2048-react-cicd-node-app-1 | Local:          http://localhost:3000/2048-in-react
2048-react-cicd-node-app-1 | On Your Network: http://172.19.0.3:3000/2048-in-react
2048-react-cicd-node-app-1 |
2048-react-cicd-node-app-1 | Note that the development build is not optimized.
2048-react-cicd-node-app-1 | To create a production build, use yarn build.
2048-react-cicd-node-app-1 |
```

Fig 3.1

#### 2. Launching Ubuntu

Instances (1) Info								
<div>Find instance by attribute or tag (case-sensitive)</div>								
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
<input type="checkbox"/>	CI-CD	i-065c10200537a1ee	Running	t2.large	2/2 checks passed	No alarms	ap-south-1a	ec2-52-66-14

Fig 3.2

### 3. Installing and running Jenkins

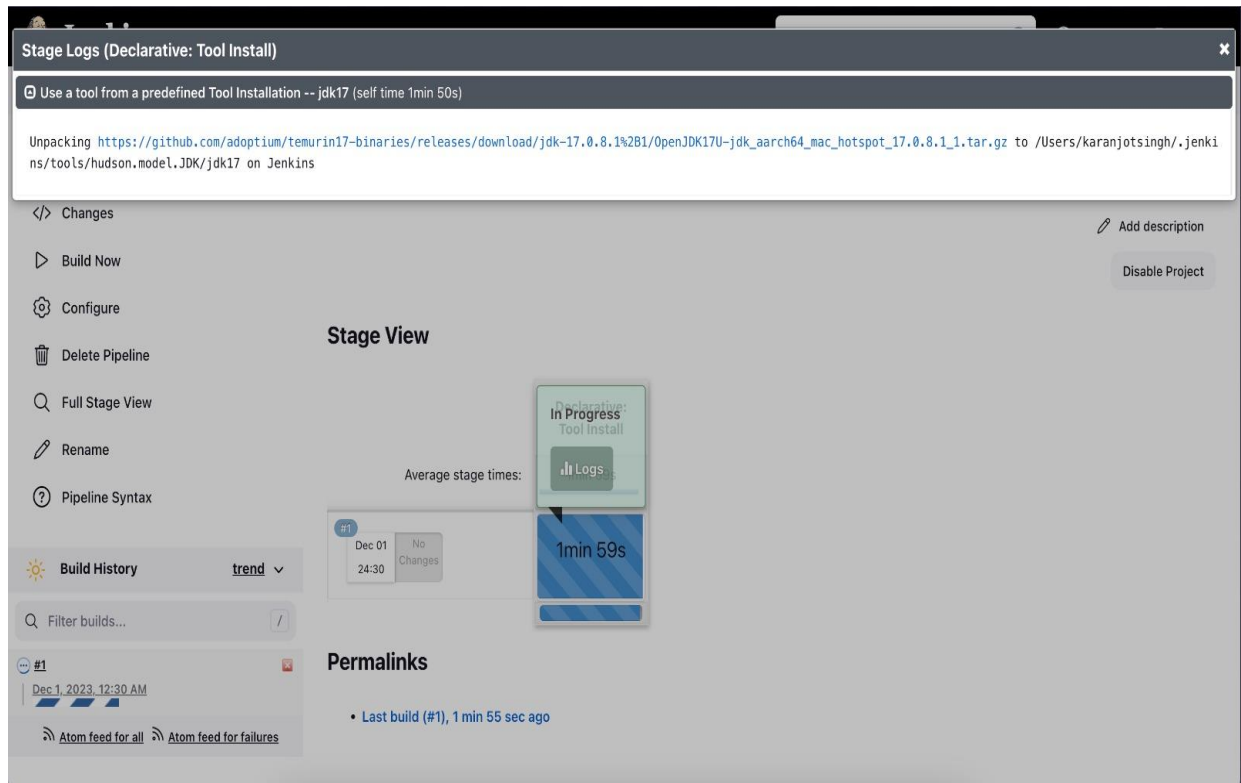


Fig 3.3

### 4. Running Docker

<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	<a href="#">mongo</a> 203fb0aeb494	latest	Unused	16 days ago	721.15 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">mongo-express</a> 5ffe220afc7a	latest	Unused	1 month ago	247.07 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">redis</a> 6f3eb785c21d	latest	In use	2 months ago	157.49 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">postgres</a> fbee27eada86	latest	In use	2 months ago	439.74 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">postgres</a> 6df5902884f9	13	In use	4 months ago	426.05 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">mailhog/mailhog</a> 4de68494cd0d	latest	In use	3 years ago	392 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">redis</a> 7a8c2384752f	4.0	In use	4 years ago	83.49 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>

Fig 3.4

## 5. Configuring Trivy

```
karanjotsingh@itsfuckindelorean>
trivy
  filesystem Scan local filesystem
  image       Scan a container image
  kubernetes  [EXPERIMENTAL] Scan kubernetes cluster
  repository  Scan a repository
  rootfs      Scan rootfs
  sbom        Scan SBOM for vulnerabilities
  vm          [EXPERIMENTAL] Scan a virtual machine image

Management Commands
  module      Manage modules
  plugin      Manage plugins

Utility Commands
  completion  Generate the autocompletion script for the specified shell
  convert     Convert Trivy JSON report into a different format
  help       Help about any command
  server      Server mode
  version     Print the version

Flags:
  -c, --cache-dir string      cache directory (default "/Users/karanjotsingh/Library/Caches/trivy")
  -c, --config string        config path (default "trivy.yaml")
  -d, --debug                debug mode
  -f, --format string        version format (json)
  --generate-default-config  write the default config to trivy-default.yaml
  -h, --help                help for trivy
  --insecure                allow insecure server connections
  -q, --quiet               suppress progress bar and log output
  --timeout duration        timeout (default 5m0s)
  -v, --version             show version

Use "trivy [command] --help" for more information about a command.
karanjotsingh@itsfuckindelorean>
trivy server
2023-12-01T00:10:25.014+0530 INFO Need to update DB
2023-12-01T00:10:25.014+0530 INFO DB Repository: ghcr.io/aquasecurity/trivy-db
2023-12-01T00:10:25.014+0530 INFO Downloading DB...
2023-12-01T00:11:11.463+0530 INFO Listening localhost:4954...
```

Fig 3.5

## 6. Install Plugins like JDK, Sonarqube Scanner, NodeJS, OWASP Dependency Check

### a) Quality Gate Waiting for Sonarqube Analysis:

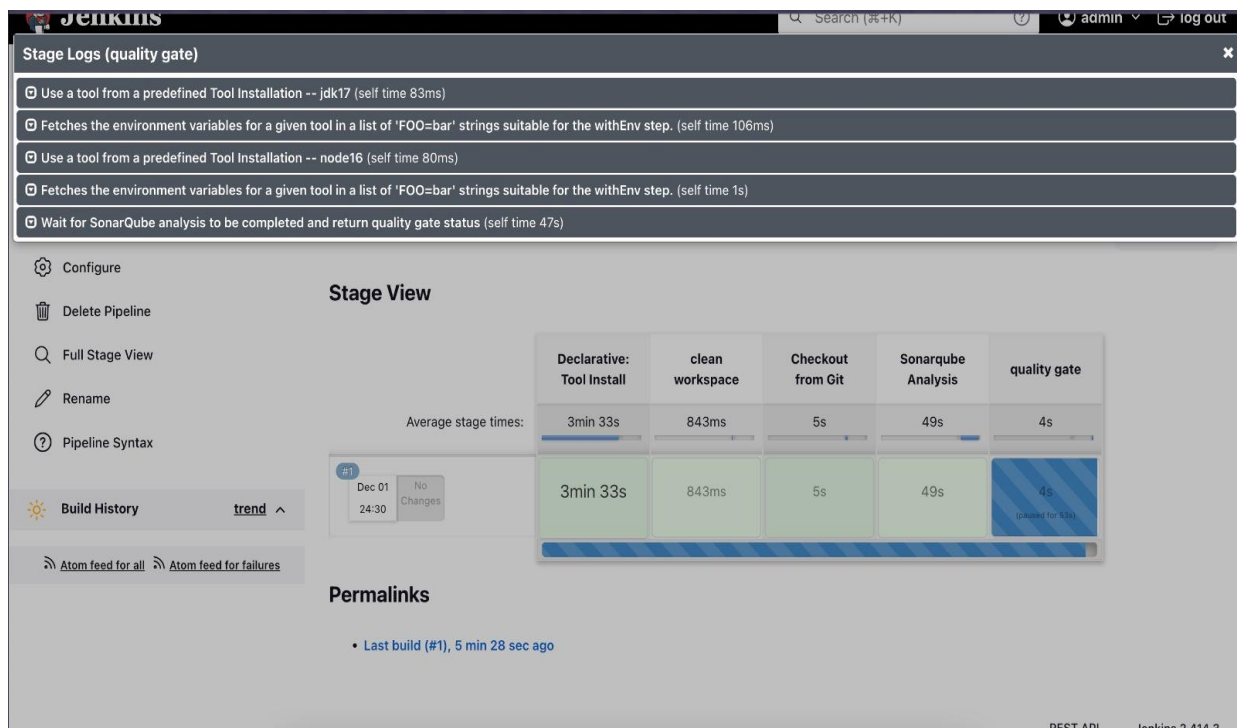


Fig 3.6

## b) Sonarqube Analysis:

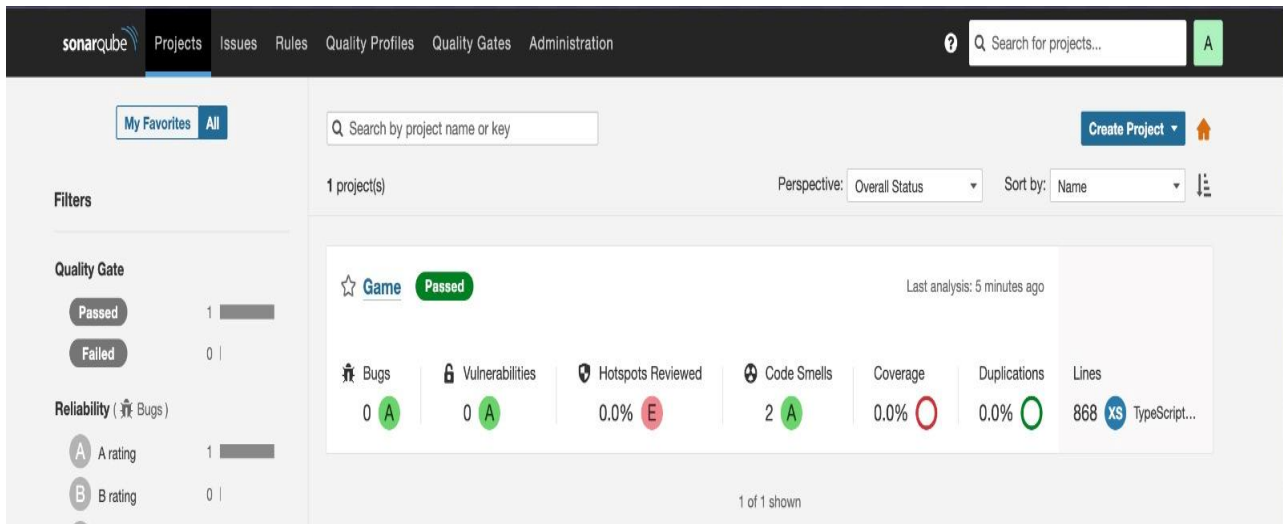


Fig 3.7

## c) OWASP Dependency Check Test:

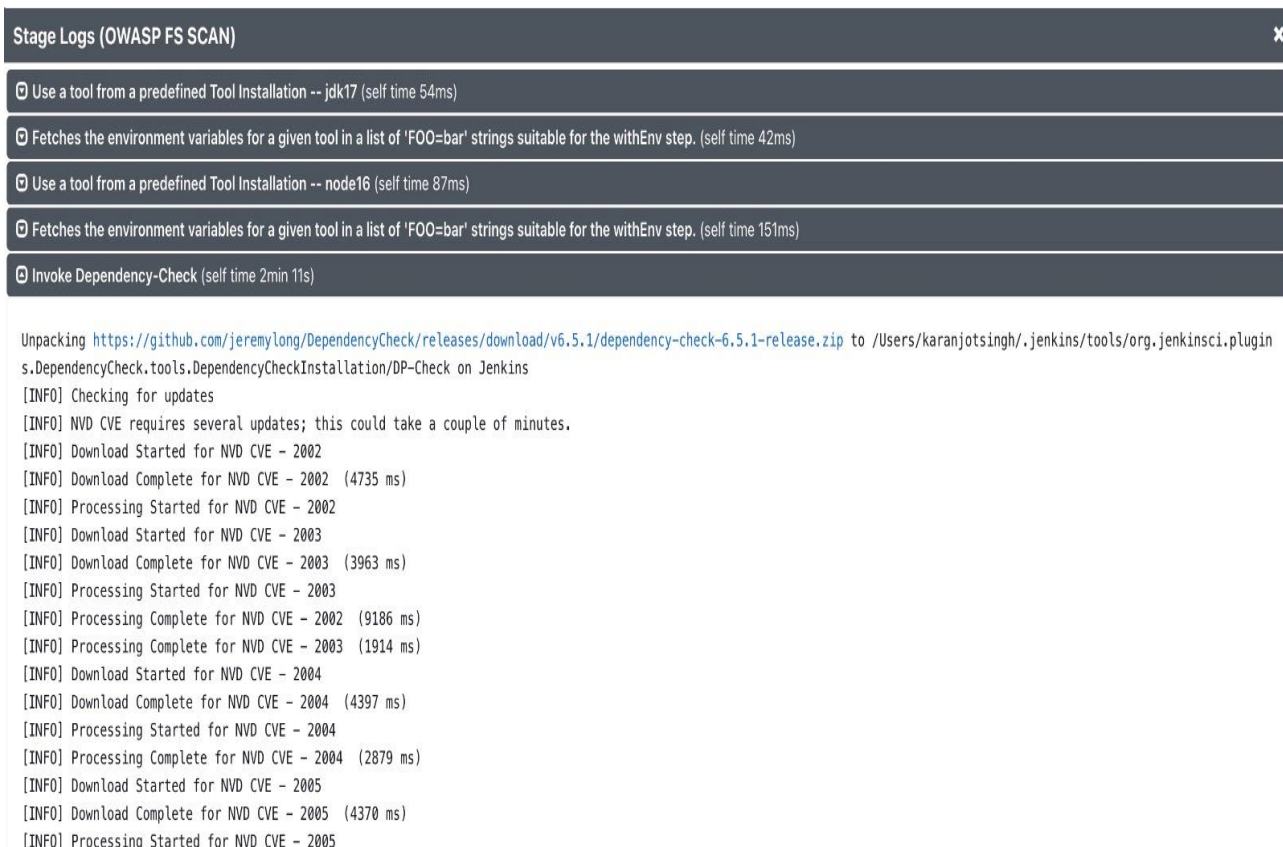


Fig 3.8

## 7. Pipeline and Add the Script in our Pipeline Script:

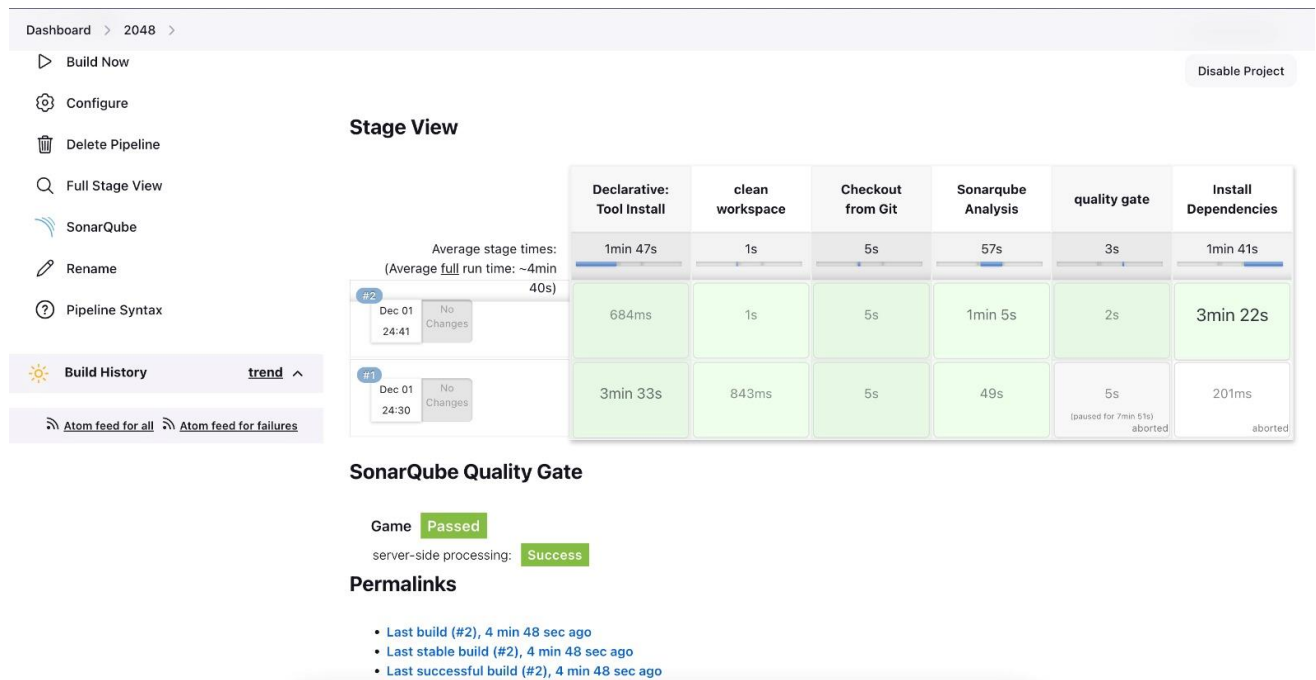


Fig 3.9

## 8. Docker Image Build, Push and Running the Application:

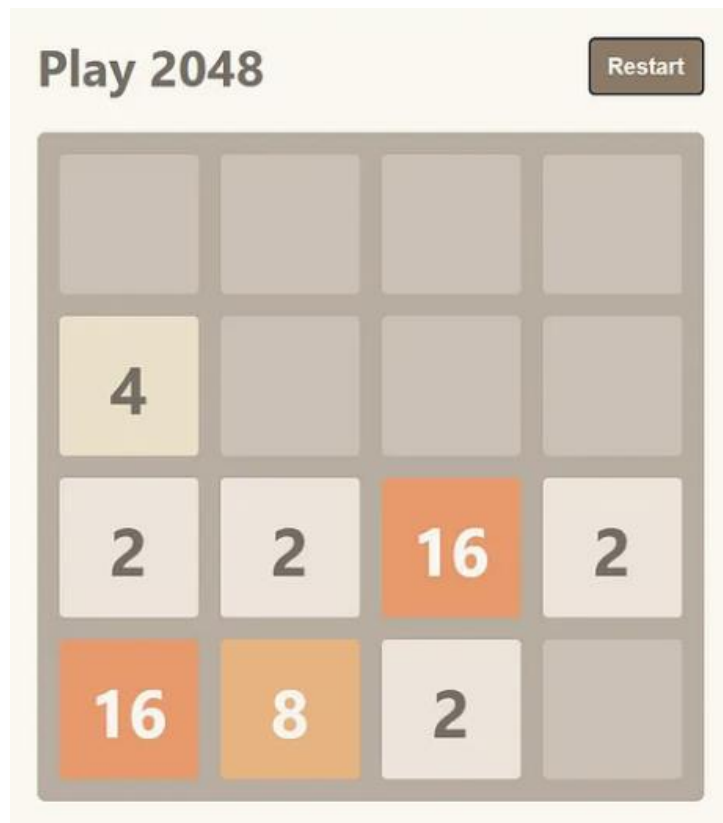


Fig 3.10

## CHAPTER- 4

### MODELLING AND IMPLEMENTATION DETAILS

#### 4.1) DESIGN DIAGRAMS

##### 1. Use Case Diagram

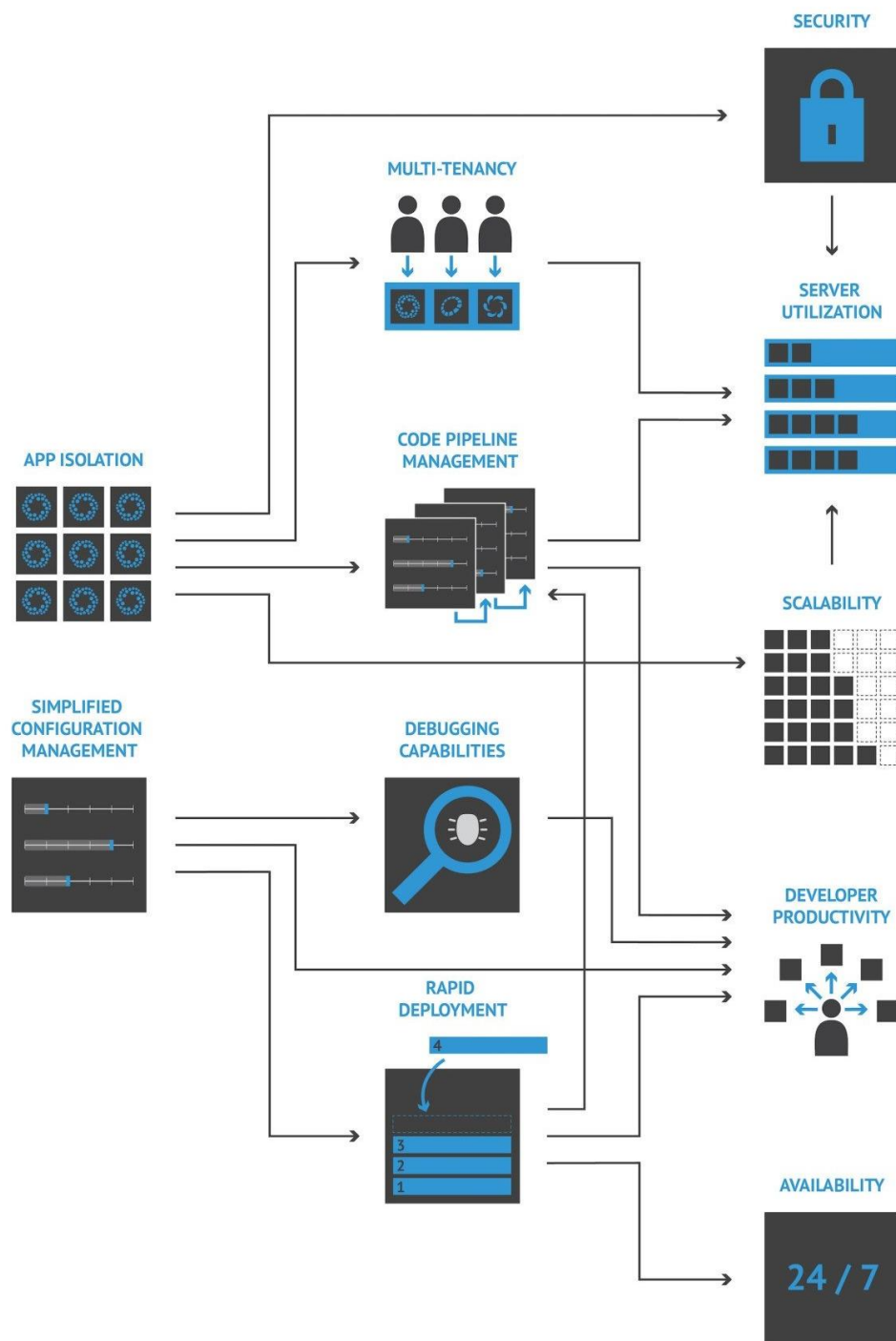


Fig 4.1



## 2. Application Design Diagram

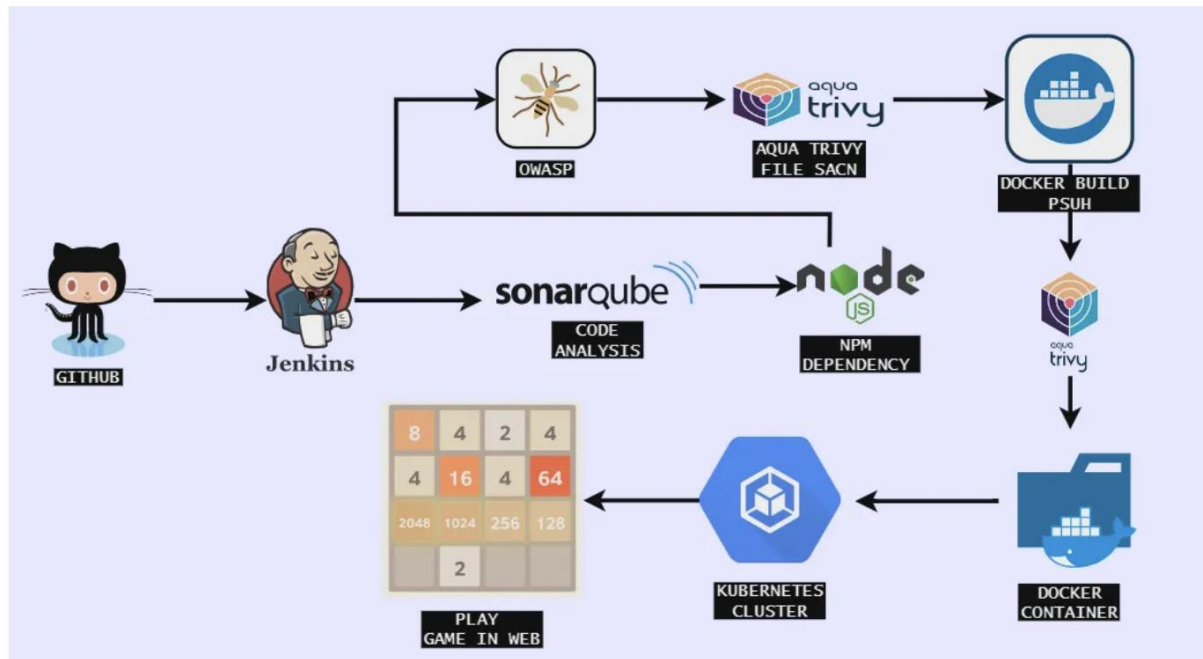


Fig 4.2

## 3. Control Flow Diagrams

### a. Using Kubernetes for deployment processes

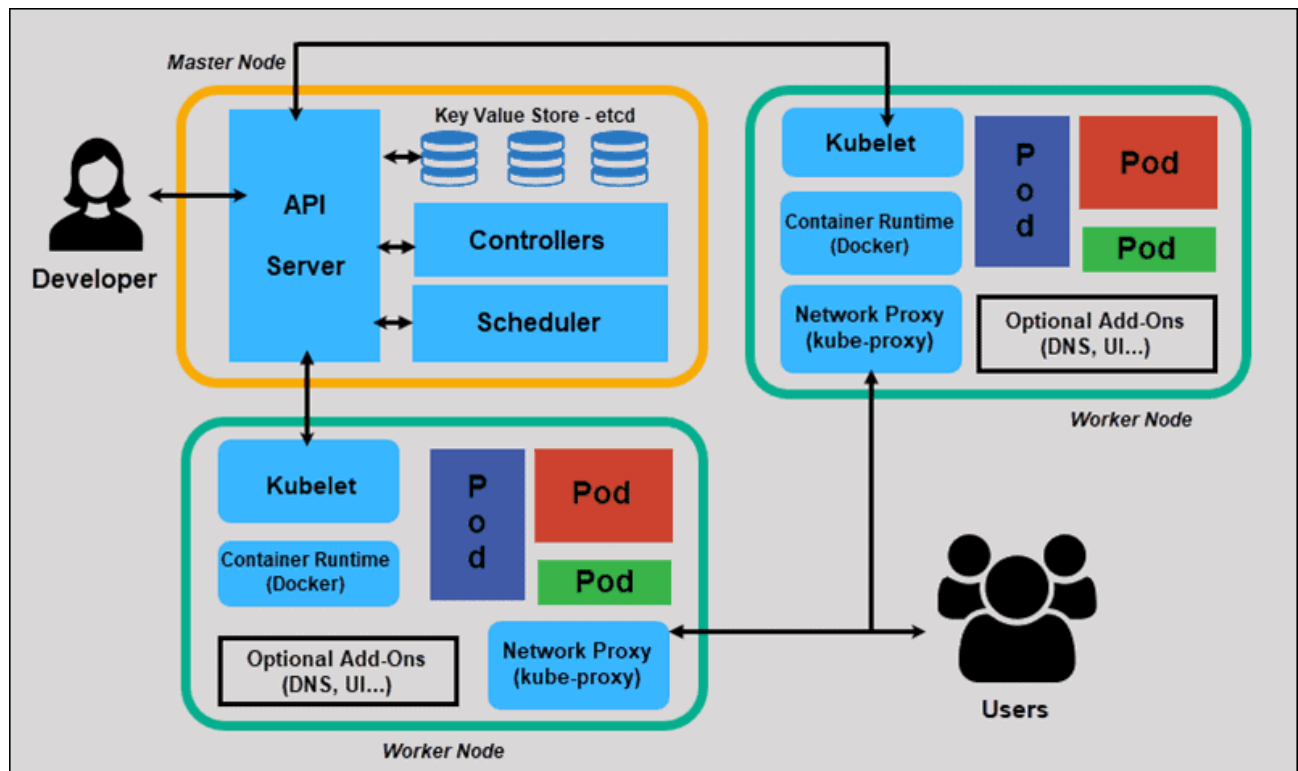


Fig 4.3



## b. Trivy for security scanning

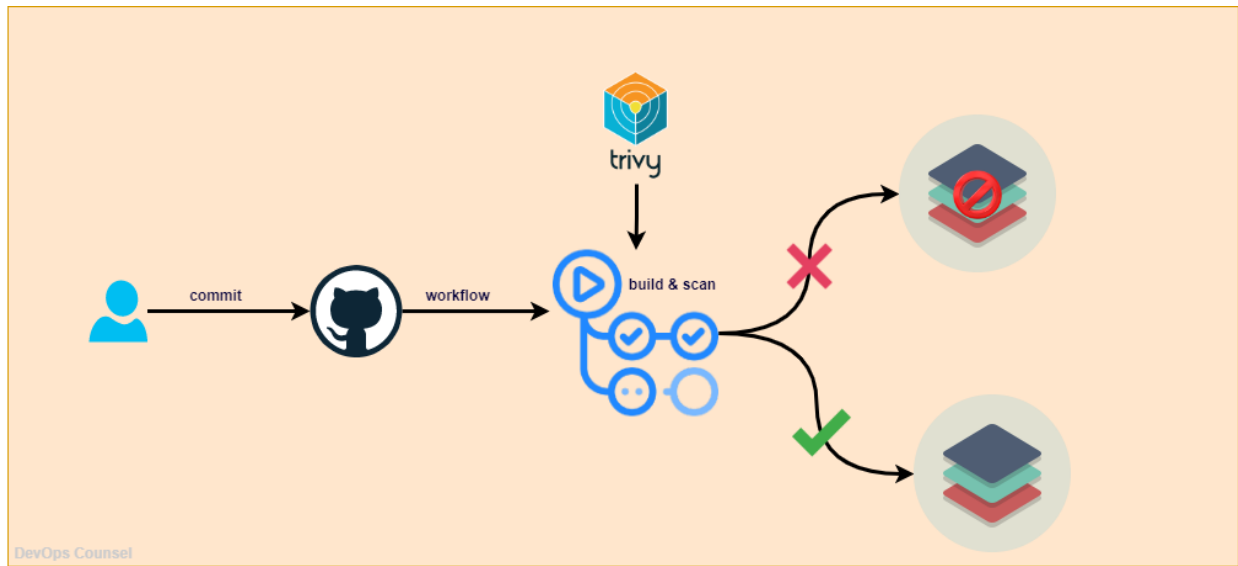


Fig 4.4

## 4. Activity Diagram

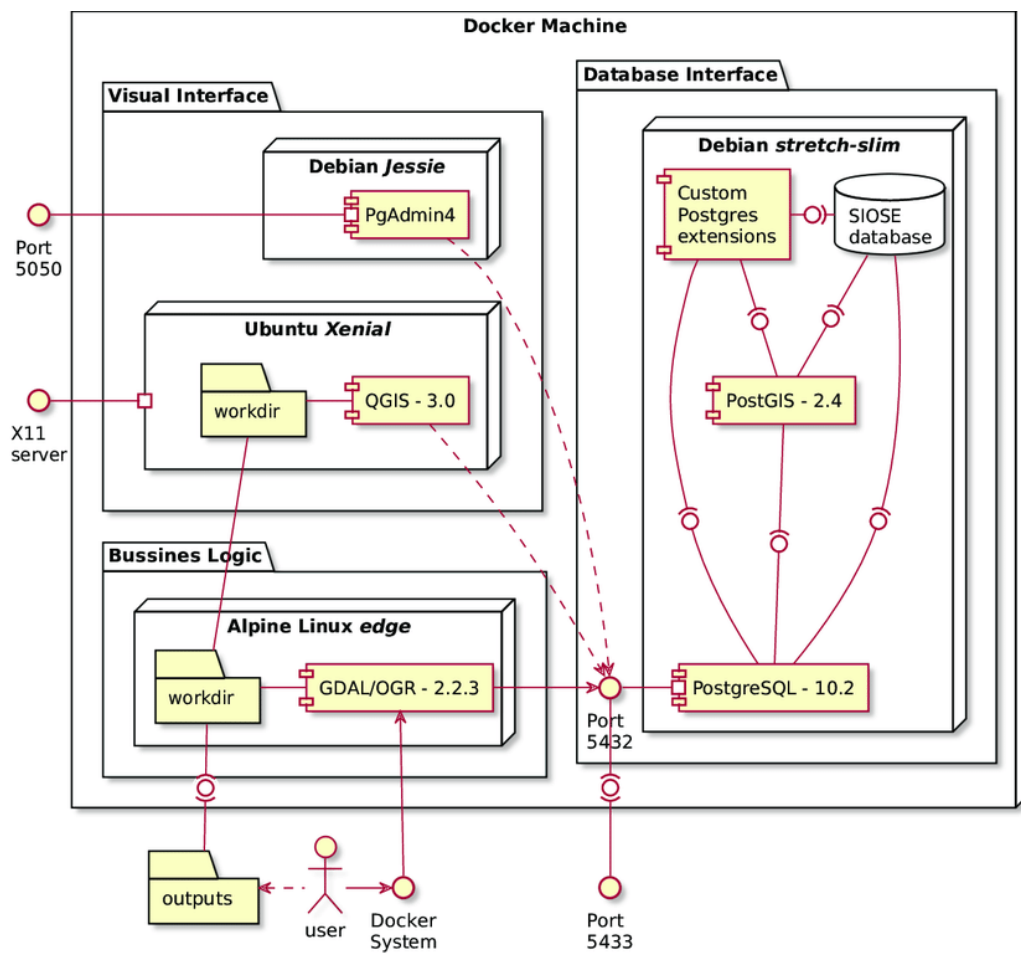


Fig 4.5

## **4.2) IMPLEMENTATION DETAILS AND ISSUES**

In our software development and deployment process, we have established a comprehensive workflow using various tools and technologies:

### **1. Jenkins as a CI/CD Tool:**

- Definition: Jenkins is an open-source CI/CD (Continuous Integration/Continuous Deployment) tool that automates software development processes.

- Role: It plays a crucial role in automating various stages of the software development lifecycle, from building and testing to deployment.

### **2. Docker for Containerization:**

- Definition: Docker is a containerization platform that simplifies application deployment and ensures consistency across different environments.

- Role: By encapsulating applications and their dependencies into containers, Docker enables seamless deployment, scalability, and portability.

### **3. Kubernetes for Container Orchestration:**

- Definition: Kubernetes is an open-source container orchestration tool designed for automating deployment, scaling, and management of containerized applications.

- Role: Kubernetes provides a robust framework for managing containerized workloads, ensuring efficient resource utilization, and simplifying the deployment and scaling of applications.

### **4. Linux Commands for System Interaction:**

- Definition: Linux commands are instructions used in the Linux operating system for system interaction and task execution.

- Role: These commands are essential for tasks such as file manipulation, user management, system monitoring, and other operations crucial for software development and deployment on a Linux environment.

## 5. JavaScript and React for Development:

- **JavaScript:** JavaScript is a versatile programming language commonly used for web development. It enables the creation of dynamic and interactive user interfaces, adding functionality to web applications.

- **React:** React is a JavaScript library for building user interfaces, particularly for creating reusable UI components. React simplifies the development of complex user interfaces, providing a structured and efficient way to manage the UI layer of web applications.

In summary, our development and deployment pipeline leverages Jenkins for automation, Docker for containerization, Kubernetes for orchestration, Linux commands for system-level operations, and JavaScript along with React for building dynamic and interactive user interfaces. This integrated approach ensures a streamlined and consistent process from development to deployment.

**Table of Commands Used:**

Commands	Descriptions
docker ps	List the running containers
docker create {image}	Create a container without starting it
docker -it {image_id}	Create an interactive shell inside container
docker rm {container}	Remove a stopped container
docker build .	Build an image from current path
docker pull {image}	Pull an image from registry
docker rmi {image}	Remove an image
kubectl get pods	Show a plain-text list of all pods
kubectl get pods --field-selector=spec.nodeName=[server-name]	Display a list of all pods running on a particular node server
kubectl create namespace [namespace-name]	To create a new namespace
kubectl apply -f [service-name].yaml	Create a new service with the definition contained in a [service-name].yaml file
kubectl describe nodes [node-name]	View details about a particular node
kubectl delete -f pod.yaml	Remove a pod using the name and type listed in pod.yaml
kubectl exec [pod-name] -- [command]	Receive output from a command run on the first container in a pod

Table 4.1

## **CHAPTER- 5**

### **TESTING**

#### **5.1) TESTING PLAN**

The goal of this load testing plan is to assess the performance, scalability, and reliability of the deployed application. Locust, a widely-used open-source load testing tool, will be employed to simulate concurrent user activity and identify potential bottlenecks and areas for optimization.

##### **1. Locust Installation:**

- Install Locust on a dedicated machine, ensuring compatibility with the deployed environment.
- Configure Locust to connect to the 2048 Game application.

##### **2. Parameterization:**

- Identify key parameters for load testing, such as the number of concurrent users, spawn rate, and test duration.
- Adjust parameters based on the expected usage patterns and system requirements.

## **5.2) COMPONENTS DECOMPOSITION AND TYPE OF TESTING REQUIRED**

### **Test Scenarios:**

#### **1. Basic User Interactions:**

- Simulate users accessing the 2048 Game, performing basic actions like starting a new game, making moves, and restarting the game.
- Evaluate response times, throughput, and system resource utilization.

#### **2. Scalability Testing:**

- Gradually increase the number of concurrent users to assess the application's scalability.
- Monitor the application's performance as the user load grows to identify scalability limits.

### **Metrics and Monitoring:**

#### **1. Response Time:**

- Measure the average response time for key user actions to ensure a responsive user experience.

#### **2. Throughput:**


- Monitor the number of requests processed per second to assess the application's throughput capabilities.

#### **3. Error Rate:**

- Track the error rate during load testing to identify potential issues and their impact on user interactions

### 5.3) LIST OF TESTS

1. Generate Comprehensive Reports: Utilize Locust's reporting features to generate detailed reports on performance metrics, including response times, throughput, and error rates.



LOCUST

HOST

<http://api.initech.com>

STATUS

RUNNING

21400 users

Edit

WORKERS

6

RPS

228.1

FAILURES

0%

STOP

Reset Stats

Statistics

Charts

Failures

Exceptions

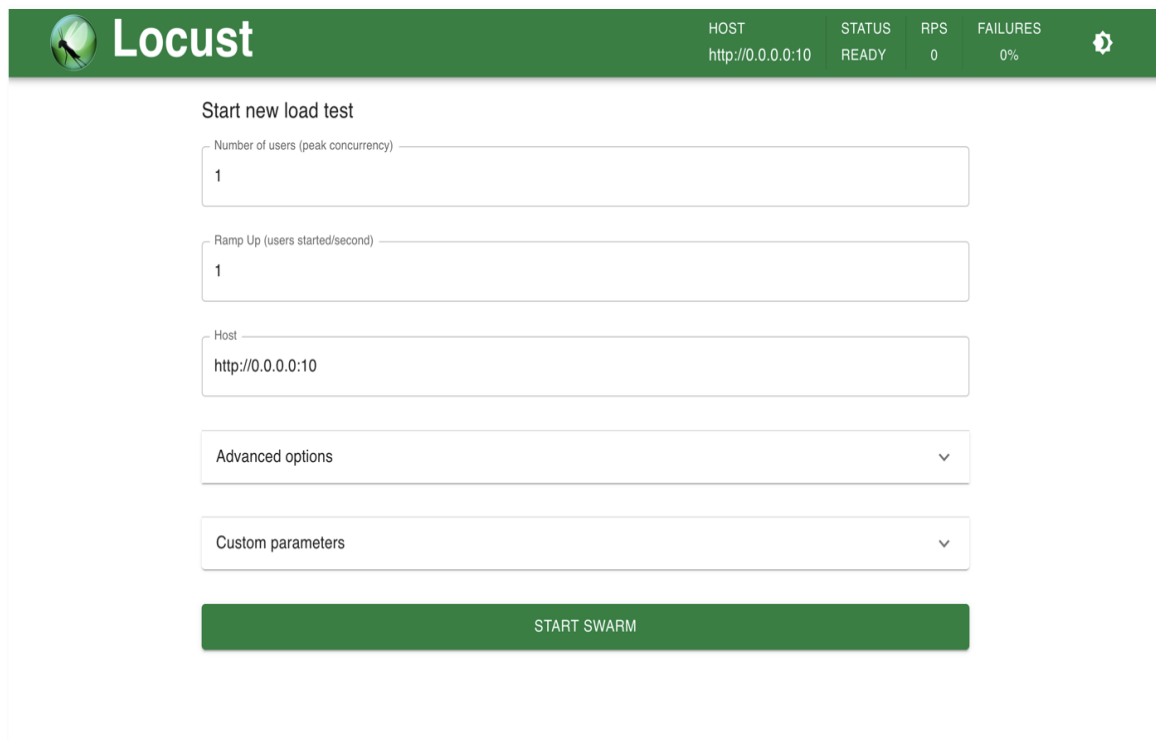
Current ratio



Download Data

Workers

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	3858	0	21	35	38	21	4	38	20170	40.1	0
GET	/blog	1279	0	25	45	49	26	3	49	20083	14.6	0
GET	/blog/[post-slug]	1258	0	14	25	27	14	2	27	20177	13.1	0
POST	/groups/create	134	0	55	100	110	58	5	109	3273	1.3	0
GET	/signin	7823	0	26	45	49	26	3	49	19969	66.3	0
POST	/signin	7823	0	83	110	120	83	45	120	20021	66.3	0

Fig 5.1




 **Locust** HOST <http://0.0.0.0:10> STATUS **READY** RPS **0** FAILURES **0%** 


Start new load test

Number of users (peak concurrency)

Ramp Up (users started/second)

Host

Advanced options 

Custom parameters 




Fig 5.2

2. Identify Bottlenecks: Analyzing test results to identify performance bottlenecks, such as slow API calls or inefficient database queries.



Fig 5.3 & Fig 5.4

3. Recommendations for Optimization: Provide recommendations for optimizations based on the identified bottlenecks, ensuring improved performance and scalability.



Fig 5.5



## 5.4) LIMITATIONS OF THE SOLUTION

The limitations we find while development and testing of the project are:

**1. Resource Intensiveness:** Running Docker containers and managing a Kubernetes cluster can be resource-intensive. It may lead to increased infrastructure costs, especially for large-scale deployments.

**2. Dependency on External Services:** The deployment relies on external services like AWS EC2 instances. Any issues with these services, such as downtime or connectivity problems, can impact the deployment process.

**3. Scalability Challenges:** While Kubernetes offers scalability, achieving optimal scalability requires careful configuration and monitoring. Inefficient scaling strategies may lead to performance issues.

**4. Limited Customization:** The standardized deployment approach may limit customization options for specific project requirements. Teams with unique needs may find it challenging to tailor the solution accordingly.

**5. Time bounded processes:** The standardized deployment approach may limit customization options, requiring additional time for teams with unique project requirements to find workarounds or alternative solutions.

Understanding and addressing these limitations during the planning and implementation phases is essential to ensure the successful deployment and ongoing management of the 2048 Game application. Regular updates, security audits, and performance optimizations can help mitigate some of these challenges over time.

## **CHAPTER- 6**

### **FINDINGS, CONCLUSION, AND FUTURE WORK**

#### **6.1) FINDINGS**

The findings from deploying the 2048 Game on Docker and Kubernetes with Jenkins CI/CD include:

- 1. Improved Deployment Efficiency:** The use of Docker and Kubernetes, orchestrated through Jenkins CI/CD, significantly improved the efficiency of the deployment process. Containerization streamlined the packaging and distribution of the game application, while Kubernetes facilitated seamless orchestration and scaling.
- 2. Enhanced Scalability:** Kubernetes demonstrated robust scalability, allowing the application to handle varying workloads efficiently. The dynamic scaling capabilities ensured optimal resource utilization and responsiveness during periods of increased user activity.
- 3. Effective CI/CD Pipeline:** Jenkins played a pivotal role in automating the continuous integration and delivery pipeline. The declarative pipeline, combined with plugins for code quality checks and vulnerability scanning, ensured a standardized and reliable deployment process.
- 4. Security Integration:** The integration of security measures, including Trivy for vulnerability scanning and OWASP Dependency Check, enhanced the overall security posture of the deployment. Identifying and addressing potential vulnerabilities during the CI/CD pipeline contributed to a more secure application.
- 5. Consistent Development Environment:** Docker provided a consistent development environment across different stages of the pipeline. Developers could build, test, and deploy the application in a standardized containerized environment, reducing compatibility issues and ensuring consistency.

**6. Accessibility and User Experience:** Accessing the game through web browsers proved to be user-friendly and accessible. The deployment allowed users to interact with the 2048 Game seamlessly, contributing to a positive user experience.

**7. Resource Optimization:** Kubernetes' autoscaling capabilities and efficient resource management contributed to optimal resource utilization. This ensured that the deployed application could scale dynamically based on demand, minimizing resource wastage.

**8. Documentation Centralization:** Establishing a centralized repository for documentation proved valuable. It served as a comprehensive reference for the deployment process, best practices, and insights. This documentation facilitated knowledge transfer and provided guidance for future development and deployments.

**9. Collaborative Development:** The use of DevOps practices, including collaborative tools like Jenkins, fostered a culture of collaboration and communication among team members. This collaborative approach contributed to smoother development and deployment workflows.

**10. Termination Process Efficiency:** The termination of AWS EC2 instances was managed efficiently, minimizing disruptions during the conclusion of the deployment cycle. Proper termination procedures ensured resource optimization and cost-effectiveness.

These findings collectively highlight the successful implementation of modern DevOps practices in deploying the application, showcasing improvements in efficiency, security, scalability, and user experience. Ongoing monitoring, optimization, and documentation efforts will further contribute to the project's success in the long term.

## 6.2) CONCLUSION

In conclusion, the deployment of the 2048 Game on Docker and Kubernetes with Jenkins CI/CD reflects a successful implementation of modern DevOps practices. The project demonstrated notable improvements in deployment efficiency, scalability, and security. Leveraging containerization through Docker, seamless orchestration with Kubernetes, and an automated CI/CD pipeline using Jenkins contributed to a streamlined and standardized deployment process. The findings underscore the positive impact on user experience, accessibility, and collaborative development. Ongoing efforts in monitoring, optimization, and centralized documentation will further enhance the project's success, emphasizing the significance of embracing DevOps methodologies for contemporary software deployment.

Moreover, the successful deployment of the 2048 Game exemplifies the adaptability of DevOps methodologies in addressing the evolving challenges of software development. The project's embrace of containerization not only streamlined deployment but also facilitated easier environment reproducibility, fostering a more consistent and reliable development environment. The seamless orchestration capabilities of Kubernetes allowed for dynamic scaling, optimizing resource utilization and ensuring the application's responsiveness to varying workloads.

Furthermore, the automated CI/CD pipeline established by Jenkins not only accelerated the release cycle but also instilled a culture of continuous integration and delivery, promoting rapid iteration and feedback loops. This approach not only enhanced development speed but also contributed to the overall quality and stability of the deployed application.

As the project continues to evolve, the focus on ongoing monitoring and optimization remains pivotal. Continuous vigilance ensures prompt identification of potential issues, allowing for proactive remediation and an overall improvement in system performance. Additionally, centralized documentation serves as a knowledge hub, fostering better collaboration, onboarding, and knowledge transfer among team members.

In essence, the deployment of the 2048 Game not only signifies a successful project but also stands as a testament to the transformative power of DevOps methodologies, fostering innovation, collaboration, and efficiency in contemporary software deployment practices.

## 6.3) FUTURE WORK

**1. Advanced Orchestration Technologies:** Explore emerging orchestration technologies beyond Kubernetes to stay at the forefront of container orchestration. Evaluate solutions that offer enhanced scalability, resource efficiency, and management capabilities.

**2. Integration of Serverless Architectures:** Investigate the integration of serverless computing models to optimize resource utilization further. Serverless architectures can offer automatic scaling and cost-efficiency for specific components of the application.

**3. Enhanced Security Measures:** Evolve the security posture by integrating advanced security measures, such as machine learning-based threat detection, enhanced access controls, and continuous monitoring. Stay abreast of evolving security standards and best practices.

**4. Adoption of Microservices Architecture:** Consider transitioning towards a microservices architecture to enhance modularity, scalability, and ease of maintenance. Decompose the application into smaller, independently deployable services for improved flexibility.

**5. Implementation of Progressive Delivery:** Explore progressive delivery practices, including canary releases and feature flagging, to gradually roll out new features and updates. This allows for real-time feedback and minimizes the impact of potential issues.

**6. Infrastructure as Code (IaC) Integration:** Implement Infrastructure as Code principles to automate the provisioning and management of infrastructure components. Tools like Terraform can enhance infrastructure reliability and repeatability.

**7. Container Security Solutions:** Integrate specialized container security solutions to fortify the overall security of the deployment. Consider tools that offer runtime protection, container image scanning, and policy enforcement.

**8. Enhanced Monitoring and Observability:** Invest in advanced monitoring and observability solutions to gain deeper insights into application performance and user behaviour. Implement distributed tracing and log analytics for comprehensive visibility.

**9. Cloud-Native Development:** Embrace cloud-native development practices and consider migrating to cloud-native services for enhanced scalability, flexibility, and cost optimization. Leverage cloud providers' managed services for databases, caching, and other components.

**10. Continuous Learning and Training:** Foster a culture of continuous learning and training for the development and operations teams. Stay updated on the latest advancements in DevOps, containerization, and cloud technologies through workshops, certifications, and industry conferences.

**11. User Feedback Mechanisms:** Implement mechanisms for collecting and analyzing user feedback to inform future feature enhancements and optimizations. Utilize user analytics and feedback loops for data-driven decision-making.

**12. Cross-Functional Collaboration:** Strengthen cross-functional collaboration between development, operations, and security teams. Encourage shared responsibilities and foster a collaborative culture to drive innovation and efficiency.

By embracing these future-oriented initiatives, the project can stay adaptive to technological advancements, enhance its capabilities, and continuously deliver value to users. The evolving landscape of DevOps and container orchestration offers ample opportunities for refinement and innovation.

## **REFERENCES**

1. <https://aquasecurity.github.io/trivy/v0.47/getting-started/installation/>
2. <https://docs.locust.io/en/stable/installation.html>
3. <https://www.docker.com/get-started/>
4. <https://www.jenkins.io/doc/>
5. <https://kubernetes.io/docs/concepts/overview/#why-you-need-kubernetes-and-what-can-it-do>
6. <https://www.npmjs.com/package/react-toastify>
7. <https://owasp.org/www-project-dependency-check/>
8.  
[https://www.researchgate.net/publication/318816158\\_An\\_Introduction\\_to\\_Docker\\_and\\_Analysis\\_of\\_its\\_Performance](https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance)
9.  
[https://www.researchgate.net/publication/303515069\\_Using\\_Docker\\_Containers\\_to\\_Improve\\_Reproducibility\\_in\\_Software\\_and\\_Web\\_Engineering\\_Research](https://www.researchgate.net/publication/303515069_Using_Docker_Containers_to_Improve_Reproducibility_in_Software_and_Web_Engineering_Research)
10.  
[https://www.researchgate.net/publication/371694855\\_DEVOPS\\_FOR\\_MANUFACTURING\\_SYSTEMS\\_SPEEDING\\_UP\\_SOFTWARE\\_DEVELOPMENT](https://www.researchgate.net/publication/371694855_DEVOPS_FOR_MANUFACTURING_SYSTEMS_SPEEDING_UP_SOFTWARE_DEVELOPMENT)
11. <https://scholar.google.com/>
12. <https://medium.com/>
13. <https://stackoverflow.com/>