

TETRIS

(Deploying and automating two versions of a Dockerized Tetris Game simultaneously with scaling using ArgoCD, Terraform, Jenkins, Kubernetes)

Enrollment Numbers: 21103105, 21103102, 21103096

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

Name of Supervisor: Dr. Amarjeet Prajapati



May – 2024

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

Part No.	Description	Page No.
II	Declaration	4
III	Certificate	5
IV	Acknowledgement	6
V	Summary	7
VI	List of Figures	8
VII	List of Symbols and Acronyms	9
VIII	List of Tables	10
Chapter No.	Topics	Page No.
Chapter-1	Introduction	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
Chapter-2	Literature Summary	19-22
	2.1 Summary of Papers Studied	19-21
	2.2 Integrated Summary of the Literature Studied	22
Chapter-3	Requirement Analysis and Solution Approach	23-32
	3.1 Overall Description of the Project	23

	3.2 Requirement Analysis	24-25
	3.3 Solution Approach	26-32
Chapter-4	Modelling and Implementation Details	33-38
	4.1 Design Diagrams	33-35
	4.2 Implementation Details and Issues	36-38
Chapter-5	Testing	39-43
	5.1 Testing Plan	39
	5.2 Components Decomposition and Type of Testing Required	40
	5.3 List of Tests	41-42
	5.4 Limitations of the Solution	43
Chapter-6	Findings, Conclusion and Future Work	44-48
	6.1 Findings	44-45
	6.2 Conclusion	46
	6.3 Future work	47-48
	REFERENCES	49

(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: 9 May, 2024

(III)

CERTIFICATE

This is to certify that the work titled **“TETRIS- Deploying and automating two versions of a Dockerized Tetris Game simultaneously with scaling using ArgoCD, Terraform, Jenkins, Kubernetes”** 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: 9 May, 2024

(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: 9 May, 2024

(V)

SUMMARY

The project embarked on a journey into the realm of DevSecOps automation, showcasing the integration of cutting-edge technologies—ArgoCD, Terraform, and Jenkins—to streamline the deployment pipeline for two versions of the beloved classic game, Tetris. With meticulous planning and collaboration, the team successfully orchestrated a seamless and secure deployment process.

The deployment pipeline began with the setup of DigitalOcean Droplets, where infrastructure provisioning and management were streamlined using Terraform. Jenkins played a pivotal role in continuous integration and delivery, orchestrating automated build processes and ensuring code quality through integration with SonarQube. ArgoCD brought declarative and GitOps-based deployment management, allowing for efficient synchronization of application changes.

Both Tetris versions were deployed with precision, leveraging the power of automation to handle infrastructure provisioning, code analysis, and Kubernetes deployment updates seamlessly. Load testing using Locust ensured the scalability and performance of the applications, providing insights into their ability to handle concurrent user requests.

Mukund Sarda

(21103105)

Priyanshu Jain

(21103102)

Karanjot Singh

(21103096)

Dr. Amarjeet Prajapati

Date: 9 May, 2024

(VI)

LIST OF FIGURES

Figure	Description	Page No.
Figure 1.1	Docker Architecture and Commands	12
Figure 3.1	Tetris Game Version-1	26
Figure 3.2	Tetris Game Version-2	26
Figure 3.3, 3.4, 3.5	AWS IAM User	27
Figure 3.6	AWSCLI	28
Figure 3.7	Jenkins Terraform	28
Figure 3.8	Jenkins	28
Figure 3.9, 3.10	Jenkins Pipeline	29
Figure 3.11	Sonarqube	29
Figure 3.12	Dependency Check	30
Figure 3.13, 3.14	Kubernetes	30, 31
Figure 3.15	ArgoCD	31
Figure 3.16, 3.17	Kubernetes Dashboard	32
Figure 4.1	Use Case Diagram	33
Figure 4.2, 4.3	Application Design Diagram, Kubernetes Deployment Diagram	34
Figure 4.4, 4.5	Trivy Security Scanning, Activity Diagram	35
Figure 5.1	Sending API Requests Using Locust	41
Figure 5.2	Configuring Locust	41
Figure 5.3	Requests Per Second Summary	42
Figure 5.4	Response Time Summary	42
Figure 5.5	Application Response Time Graph W.R.T Other API Requests	42

(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

Table	Description	Page No.
Table 4.1	Commands Used	36

CHAPTER- 1

INTRODUCTION

1.1) GENERAL INTRODUCTION

In today's rapidly evolving software landscape, the need for secure, efficient, and automated deployment practices has never been greater. DevSecOps, a philosophy that integrates security practices seamlessly into the DevOps pipeline, represents a fundamental shift in how organizations approach software development and deployment. By prioritizing security from the outset and embedding it throughout the development lifecycle, DevSecOps aims to minimize vulnerabilities, reduce risks, and enhance the overall resilience of deployed applications.

Our project embodies the principles of DevSecOps as we endeavor to optimize the deployment process of Tetris applications. At the core of our approach lies a suite of automation tools, each serving a distinct purpose in orchestrating a secure and efficient deployment pipeline.

ArgoCD, a GitOps continuous delivery tool, empowers us to automate the deployment of Tetris applications by providing declarative and Git-centric workflows. With ArgoCD, we can ensure consistency and reliability in deploying multiple game versions while adhering to security best practices.

Terraform, an Infrastructure as Code (IaC) tool, enables us to provision and manage infrastructure resources programmatically. Through Terraform, we automate the deployment of DigitalOcean resources, ensuring scalability, repeatability, and security in our infrastructure setup.

Jenkins, a leading automation server, plays a pivotal role in our CI/CD pipeline, automating the building, testing, and deployment of Tetris applications. By integrating security testing and code analysis into our Jenkins pipeline, we mitigate risks and ensure the robustness of our deployments.

In this report, we delve into the intricate workings of DevSecOps and its application in modern software development. Through a detailed analysis of our project's methodology, utilization of automation tools, and adherence to security practices, we aim to showcase the transformative potential of DevSecOps in optimizing deployment processes and enhancing the security posture of deployed applications.

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.

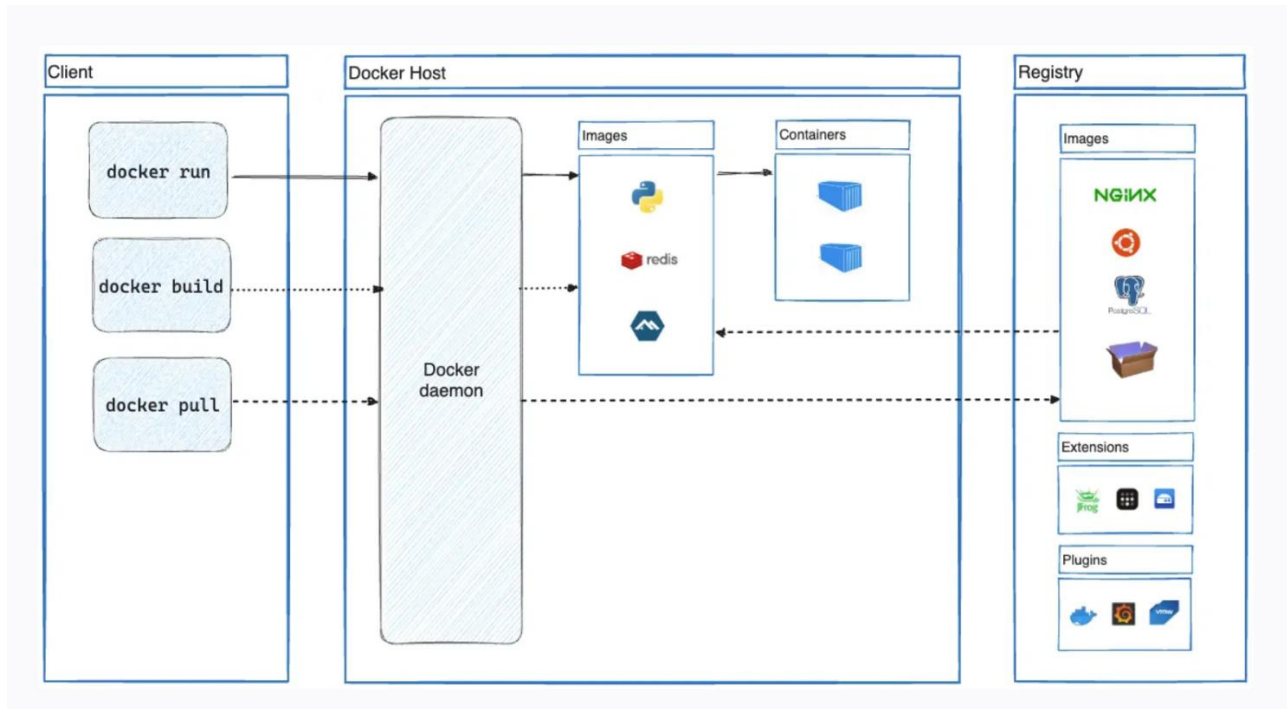


Fig 1.1

1.2) PROBLEM STATEMENT

In traditional software deployment practices, the process of deploying and managing applications often lacks efficiency, scalability, and security. Manual interventions, inconsistent configurations, and a fragmented approach to security testing can result in deployment bottlenecks, increased risk exposure, and compromised application integrity.

The problem statement addressed by our project revolves around the need for a streamlined, secure, and automated deployment pipeline for Tetris applications. Specifically, we aim to tackle the following challenges:

1. **Manual Deployment Processes:** Traditional deployment processes often involve manual interventions, leading to inefficiencies, errors, and delays in application deployment. Automating these processes is essential to improve deployment speed, consistency, and reliability.
2. **Security Vulnerabilities:** Inadequate security measures and fragmented security testing processes leave applications vulnerable to cyber threats and breaches. Integrating security practices seamlessly into the deployment pipeline is crucial to mitigate risks and ensure the security of deployed applications.
3. **Scalability and Consistency:** Managing infrastructure resources manually can be cumbersome and error-prone, particularly in environments with multiple application versions and deployments. Implementing infrastructure as code (IaC) practices is essential to ensure scalability, repeatability, and consistency in infrastructure provisioning.
4. **Lack of Continuous Integration and Delivery (CI/CD):** Traditional deployment approaches often lack robust CI/CD pipelines, leading to manual and time-consuming release processes. Implementing CI/CD practices enables automated building, testing, and deployment of applications, resulting in faster delivery cycles and improved overall efficiency.

By addressing these challenges through the implementation of DevSecOps principles and leveraging automation tools like ArgoCD, Terraform, and Jenkins, our project seeks to revolutionize the deployment process of Tetris applications. Through automation, integration, and security-centric practices, we aim to create a deployment pipeline that is efficient, scalable, and resilient, ensuring the seamless deployment of Tetris applications while upholding the highest standards of security

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. **Efficiency and Productivity Gains:** Streamlining the deployment process through automation reduces manual effort and accelerates application deployment. This efficiency gain translates into faster time-to-market for new features and updates, enhancing overall productivity and competitiveness in the market.
2. **Enhanced Security Posture:** Integrating security practices into the deployment pipeline mitigates the risk of security breaches and data leaks. By automating security testing and ensuring adherence to security best practices, organizations can bolster their security posture and safeguard sensitive information from potential threats.
3. **Improved Scalability and Reliability:** Implementing infrastructure as code (IaC) practices enables organizations to provision and manage infrastructure resources programmatically, ensuring scalability, repeatability, and consistency. This scalability and reliability lay the foundation for handling increased workload demands and maintaining application performance under varying conditions.
4. **Faster Delivery Cycles:** Adopting continuous integration and delivery (CI/CD) practices streamlines the release process, allowing organizations to deliver new features and updates to users more frequently and reliably. This accelerated delivery cycle enhances customer satisfaction, fosters innovation, and keeps pace with rapidly evolving market demands.
5. **Cost Savings:** Automation reduces the overhead associated with manual deployment processes, resulting in cost savings through improved resource utilization and reduced operational expenses. By optimizing resource allocation and minimizing downtime, organizations can achieve significant cost efficiencies while maximizing return on investment.

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. **Infrastructure Provisioning:** Begin by setting up the infrastructure using Terraform, configuring DigitalOcean resources, and establishing the necessary networking components to support the deployment pipeline.
2. **Toolchain Installation:** Install and configure essential tools including Jenkins, SonarQube, Trivy, DigitalOcean CLI, Kubectl, and Terraform to facilitate development, testing, and deployment processes.
3. **Pipeline Development:** Develop a comprehensive CI/CD pipeline in Jenkins to automate build, test, and deployment tasks, ensuring consistency and efficiency in the software delivery process.
4. **Security Integration:** Integrate security measures by incorporating SonarQube for code analysis, Trivy for vulnerability scanning, and OWASP Dependency Check for identifying and mitigating security vulnerabilities.
5. **Containerization and Deployment:** Build Docker images of the Tetris applications, push them to Docker Hub, and deploy them using Kubernetes for container orchestration, ensuring scalability and high availability.
6. **GitOps Implementation:** Implement Argo CD as a GitOps continuous delivery tool to automate application deployment and management on Kubernetes clusters, providing declarative and version-controlled deployments.
7. **User Accessibility Enhancement:** Enhance user accessibility by enabling access to the Tetris applications through web browsers, ensuring seamless interaction and engagement.
8. **Resource Management:** Conclude the deployment process by efficiently managing resources, including terminating DigitalOcean EC2 instances and optimizing cloud usage to reduce costs and improve resource utilization

3.2) REQUIREMENT ANALYSIS

1. Infrastructure Requirements:

- Provision DigitalOcean Droplets with appropriate configurations for the Tetris application deployment.
- Set up a Kubernetes Cluster using DigitalOcean Kubernetes (DOKS) for container orchestration.
- Configure networking to facilitate communication between components and external access.

2. Software and Tools:

- Install and configure Argo CD for GitOps-based deployment management.
- Utilize Terraform for infrastructure provisioning and management.
- Implement Jenkins for CI/CD pipeline orchestration.
- Incorporate Docker for containerization of the Tetris application.

3. Development and Deployment Dependencies:

- Utilize React for developing the Tetris application frontend.
- Ensure compatibility with modern web browsers for user access.

4. Jenkins Pipeline Configuration:

- Establish Jenkins pipeline jobs for building, testing, and deploying Tetris application updates.
- Integrate Terraform and Kubernetes plugins for infrastructure provisioning and deployment.

5. GitOps Workflow Integration:

- Configure Argo CD to synchronize application state with Git repositories, enabling declarative and automated deployments.
- Implement a GitOps workflow for managing Tetris application updates through Git commits.

6. Security Measures:

- Implement RBAC (Role-Based Access Control) in Kubernetes to control access to resources.
- Secure sensitive credentials and configurations using Kubernetes Secrets and Argo CD encryption features.

7. Scalability and Resource Optimization:

- Configure Horizontal Pod Autoscaling (HPA) in Kubernetes for dynamic scaling based on resource utilization.

- Optimize resource allocation and utilization to ensure efficient Tetris application performance.

8. Monitoring and Observability:

- Set up Prometheus and Grafana for monitoring and visualizing cluster metrics and Tetris application performance.

- Implement logging with tools like Fluentd and Elasticsearch for tracking application events and debugging.

9. Documentation and Knowledge Transfer:

- Establish comprehensive documentation detailing setup instructions, configuration guides, and troubleshooting steps.

- Conduct knowledge transfer sessions to familiarize team members with the project architecture and deployment process.

10. User Acceptance Testing (UAT):

- Develop UAT scenarios and test cases to validate Tetris application functionality, performance, and user experience.

- Utilize staging environments for conducting UAT before promoting changes to production.

3.3) SOLUTION APPROACH

1. Tetris Game Version-1

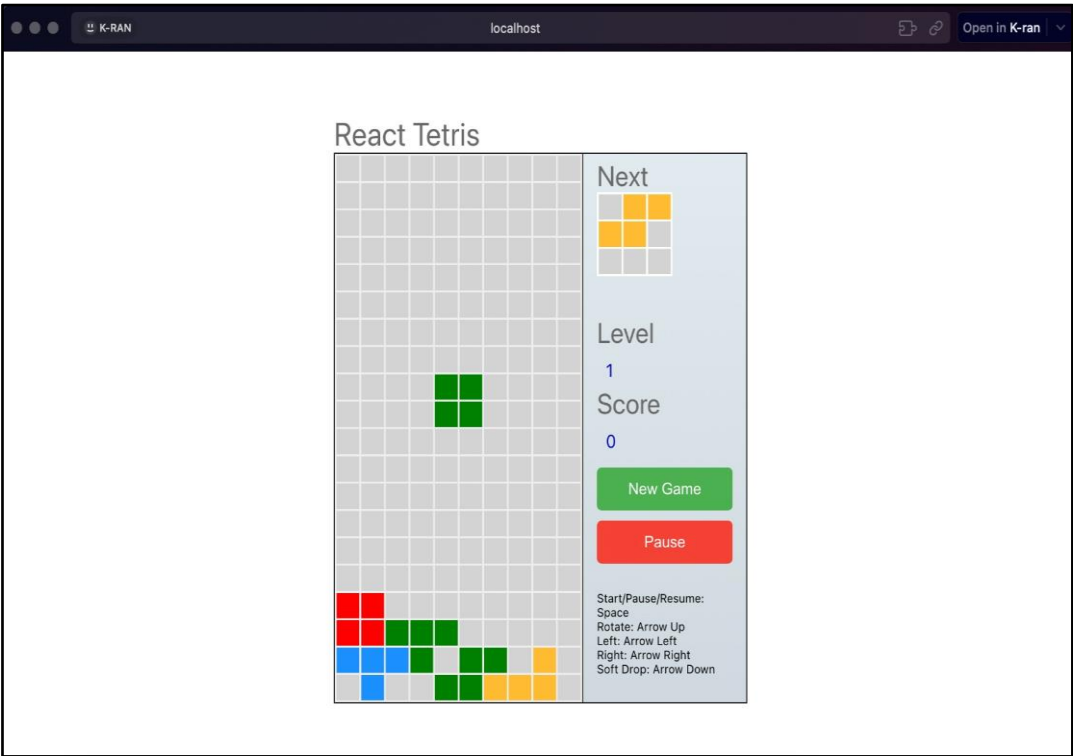


Fig 3.1

2. Tetris Game Version-2

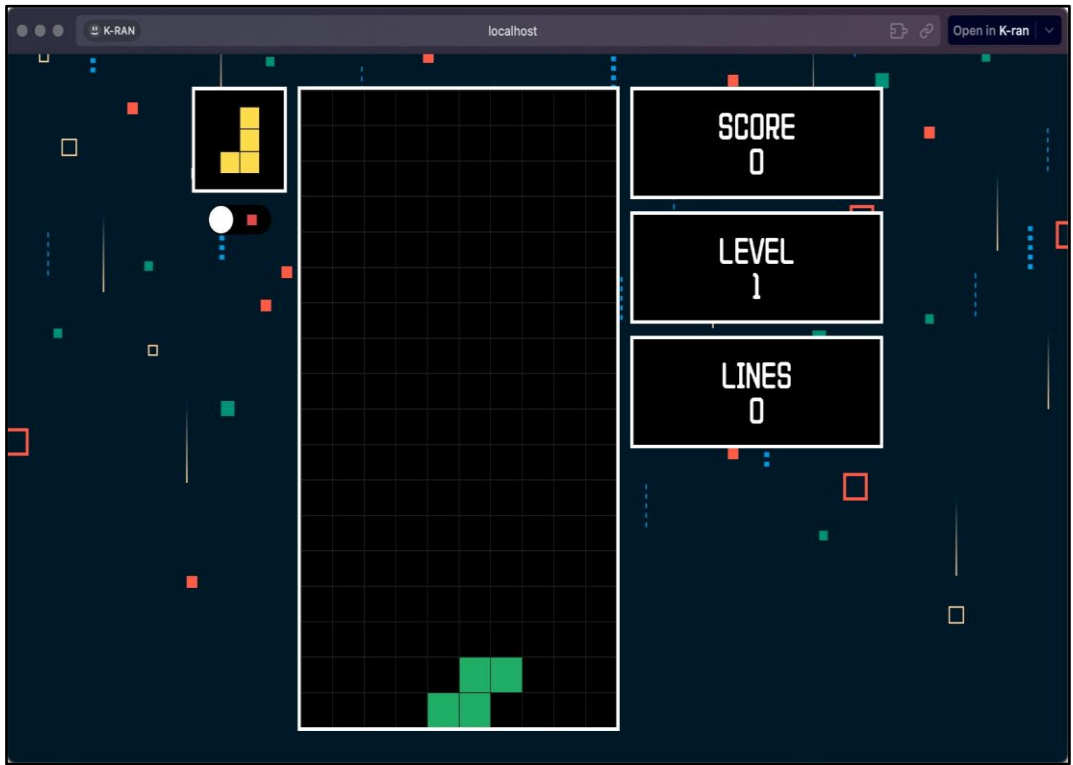


Fig 3.2

3. AWS IAM User

The screenshot shows the 'Specify user details' step of the AWS IAM 'Create user' process. The breadcrumb navigation is 'IAM > Users > Create user'. The left sidebar shows three steps: 'Step 1: Specify user details' (active), 'Step 2: Set permissions', and 'Step 3: Review and create'. The main content area is titled 'Specify user details' and contains a 'User details' section. In this section, the 'User name' field is populated with 'Karanjot-Minor-Project'. Below the field, a note states: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)'. There is an unchecked checkbox for 'Provide user access to the AWS Management Console - optional' with a note: 'If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.' Below this is a blue information box with a circular icon containing an 'i' and the text: 'If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)'. At the bottom right of the form are 'Cancel' and 'Next' buttons.

Fig 3.3

The screenshot shows the 'Permissions policies' selection screen in the AWS IAM console. The title is 'Permissions policies (1/1196)'. Below the title is the instruction 'Choose one or more policies to attach to your new user.' There are two buttons at the top right: a refresh button and a 'Create policy' button with an external link icon. A search bar with the placeholder 'Search' is on the left. To its right is a 'Filter by Type' dropdown menu currently set to 'All types'. Below these is a table with three columns: 'Policy name', 'Type', and 'Attached entities'. The table contains three rows of policies. The first row is 'AccessAnalyzerServiceRolePolicy' (AWS managed) with 0 attached entities. The second row, 'AdministratorAccess' (AWS managed - job function), is selected with a blue background and a checked checkbox. The third row is 'AdministratorAccess-Amplify' (AWS managed) with 0 attached entities. Each row has a checkbox on the left and an external link icon next to the policy name.

	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerServiceRolePolicy	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	0
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	0

Fig 3.4

The screenshot shows the 'Console sign-in details' screen in the AWS IAM console. The title is 'Console sign-in details'. At the top right is a button 'Email sign-in instructions' with an external link icon. The main content area displays three fields: 'Console sign-in URL' with the value 'https://094403868479.signin.aws.amazon.com/console', 'User name' with the value 'Karanjot-Minor-Project', and 'Console password' with a masked value '*****' and a 'Show' link. At the bottom right are three buttons: 'Cancel', 'Download .csv file', and 'Return to users list'.

Fig 3.5

4. AWSCLI

```
~/Tetris-gamev1 git:(main)±1 (16.975s)
aws configure
AWS Access Key ID [*****RLJZ]: AKIARL6XIW47VDBGFVUG
AWS Secret Access Key [*****BxIB]: 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
Default region name [ap-south-1]:
Default output format [json]:
```

Fig 3.6

5. Jenkins Terraform

```
~/Tetris-gamev1/Jenkins-terraform git:(main)±1 (0.035s)
cat Main.tf
resource "aws_iam_role" "example_role" {
  name = "Jenkins-terraform"
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
}

resource "aws_iam_role_policy_attachment" "example_attachment" {
  role       = aws_iam_role.example_role.name
  policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
}

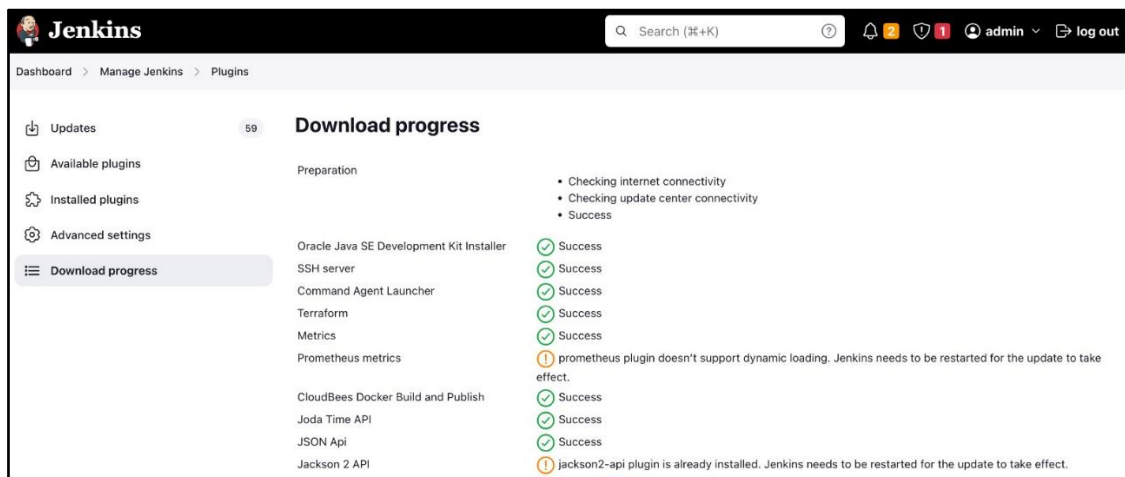
resource "aws_iam_instance_profile" "example_profile" {
  name = "Jenkins-terraform"
  role = aws_iam_role.example_role.name
}

resource "aws_security_group" "Jenkins-sg" {
  name        = "Jenkins-Security Group"
  description = "Open 22,443,80,8080,9000"

  # Define a single ingress rule to allow traffic on all specified ports
  ingress = [
    for port in [22, 80, 443, 8080, 9000, 3000] : {
      description = "TLS from VPC"
    }
  ]
}
```

Fig 3.7

6. Jenkins



The screenshot shows the Jenkins web interface with the 'Download progress' page selected. The page displays a list of plugins and their installation status. The 'Updates' section shows 59 updates. The 'Download progress' section lists the following plugins and their status:

Plugin	Status
Preparation	Success
Oracle Java SE Development Kit Installer	Success
SSH server	Success
Command Agent Launcher	Success
Terraform	Success
Metrics	Success
Prometheus metrics	Warning: prometheus plugin doesn't support dynamic loading. Jenkins needs to be restarted for the update to take effect.
CloudBees Docker Build and Publish	Success
Joda Time API	Success
JSON Api	Success
Jackson 2 API	Warning: jackson2-api plugin is already installed. Jenkins needs to be restarted for the update to take effect.

Fig 3.8

7. Jenkins Pipeline

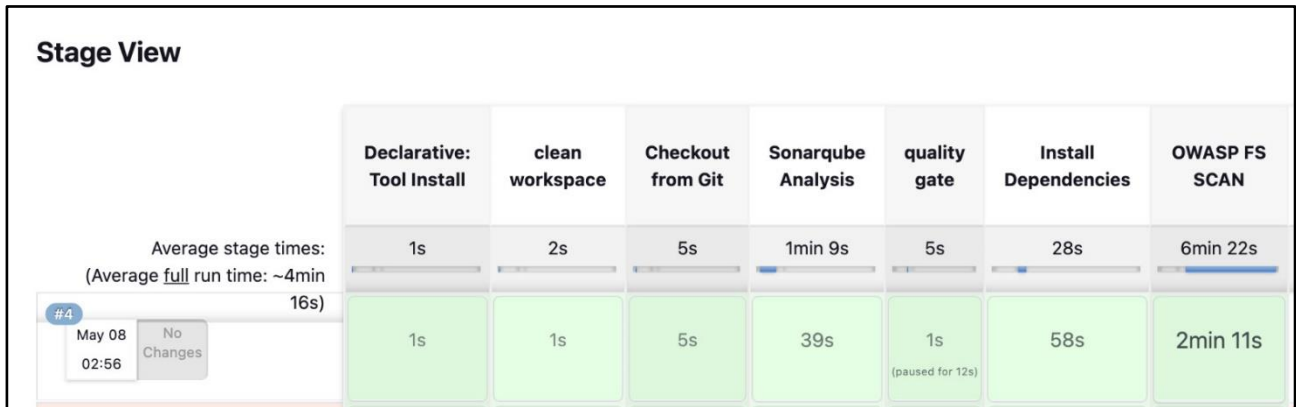


Fig 3.9

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☁	Tetris v1 Image Build	7 hr 10 min #4	7 hr 39 min #3	4 min 16 sec ▶
✗	☁	Tetris v2 Image Build	N/A	6 hr 12 min #10	8 min 27 sec ▶

Fig 3.10

8. Sonarqube

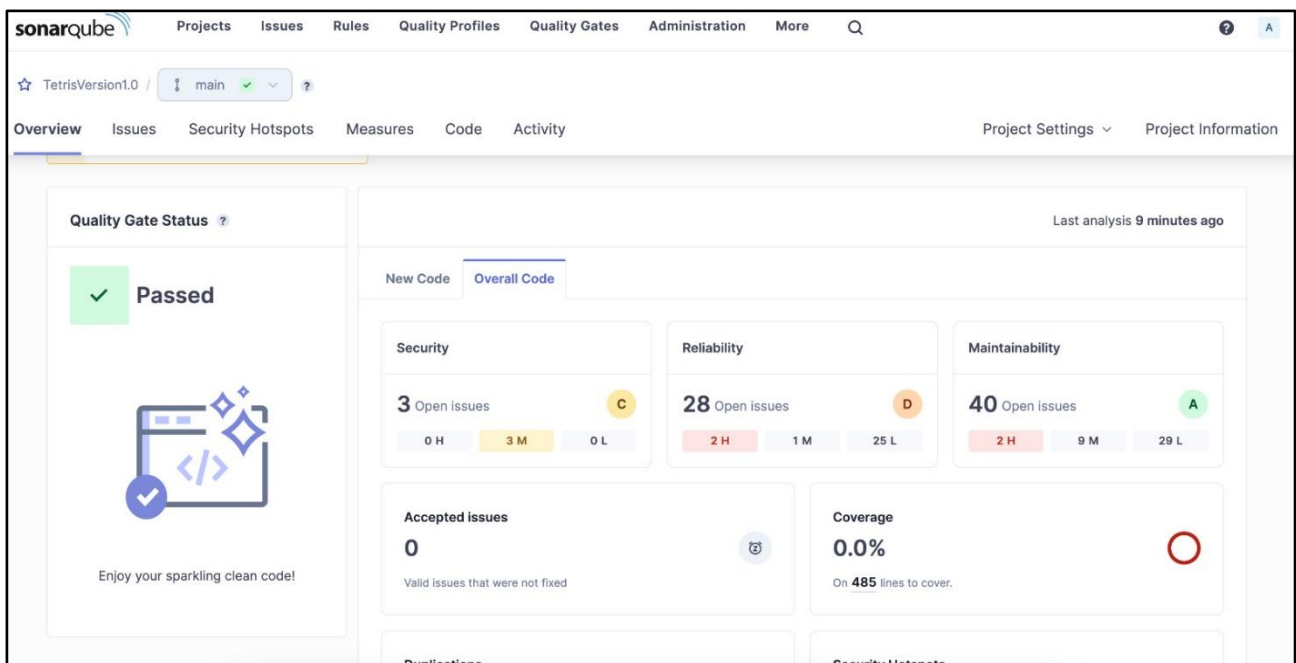


Fig 3.11

9. Dependency Check

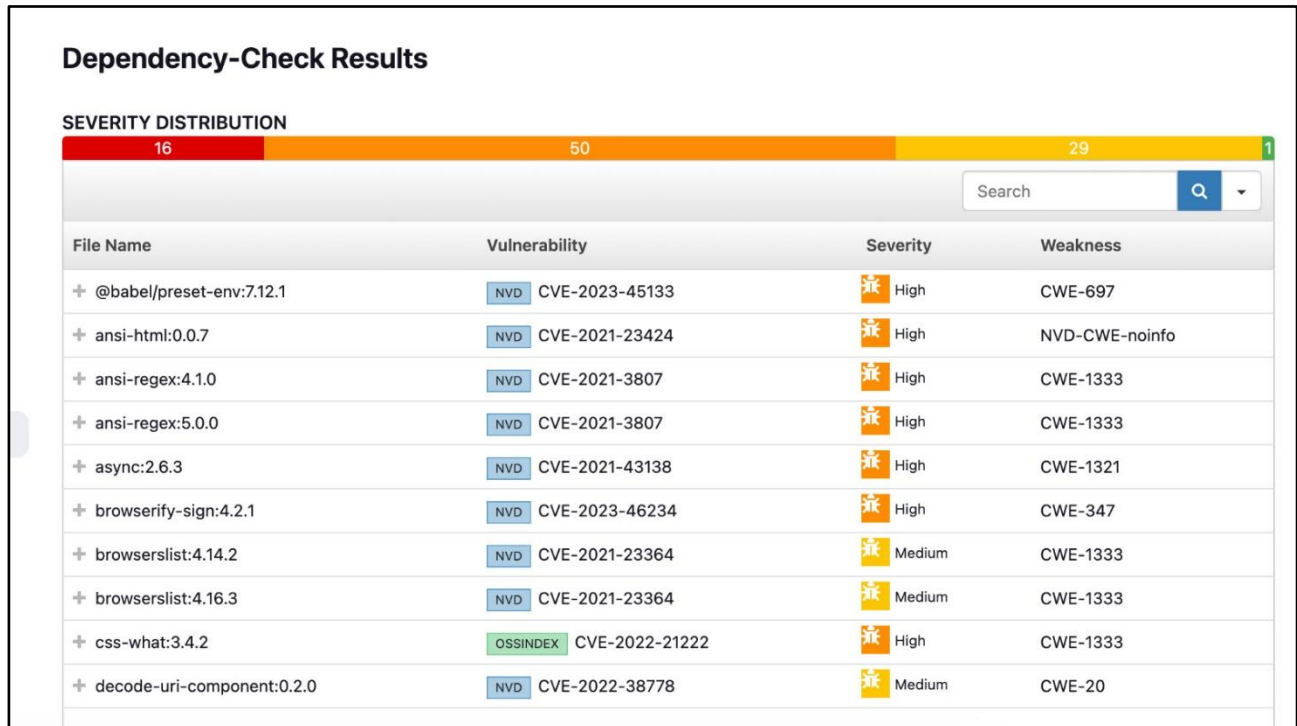


Fig 3.12

10. Kubernetes

```
~/minor-project-2/Tetris-gamev1 git:(main)±3 * do-nyc1-tetrisv1 (1.389s)
kubect1 scale deployment/tetrisv1 --replicas=20

deployment.apps/tetrisv1 scaled

~/minor-project-2/Tetris-gamev1 git:(main)±3 * do-nyc1-tetrisv1 (1.019s)
kubect1 get rs
NAME                                DESIRED    CURRENT    READY    AGE
tetrisv1-5fd57d657d                 20         20         0        6m46s

~/minor-project-2/Tetris-gamev1 git:(main)±3 * do-nyc1-tetrisv1 (1.562s)
kubect1 get pods
NAME                                READY     STATUS                    RESTARTS   AGE
tetrisv1-5fd57d657d-2zf27           0/1       Error                     0           14s
tetrisv1-5fd57d657d-4px8c           1/1       Running                   1 (3s ago)  14s
tetrisv1-5fd57d657d-64m88           0/1       Error                     1 (6s ago)  14s
tetrisv1-5fd57d657d-65pbj           0/1       Error                     1 (6s ago)  14s
tetrisv1-5fd57d657d-6gl92           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-84z9d           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-9fxcc           0/1       CrashLoopBackOff          1 (3s ago)  14s
tetrisv1-5fd57d657d-cmbgw           0/1       Error                     1 (7s ago)  14s
tetrisv1-5fd57d657d-fjf4m           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-gfw2h           0/1       Error                     1 (6s ago)  14s
tetrisv1-5fd57d657d-ggngx           0/1       CrashLoopBackOff          1 (3s ago)  14s
tetrisv1-5fd57d657d-jscxw           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-nbhm5           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-pl856           0/1       CrashLoopBackOff          1 (2s ago)  14s
tetrisv1-5fd57d657d-q79mn           0/1       CrashLoopBackOff          1 (2s ago)  14s
tetrisv1-5fd57d657d-wfm26           1/1       Running                   1 (6s ago)  14s
tetrisv1-5fd57d657d-xc4sv           1/1       Running                   0           14s
tetrisv1-5fd57d657d-xnnjc           1/1       Running                   1 (5s ago)  14s
tetrisv1-5fd57d657d-z8qnb           0/1       ContainerCreating         0           14s
tetrisv1-5fd57d657d-zl26f           0/1       CrashLoopBackOff          6 (19s ago) 6m54s
```

Fig 3.13

```
~/minor-project-2/Tetris-gamev1 git:(main) #3 * do-nyc1-tetrisv1 (1.532s)
kubectl get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name --all-namespaces | grep tetrisv1
mypool-j4y1t tetrisv1-5fd57d657d-2zf27
mypool-j4y1t tetrisv1-5fd57d657d-4px8c
mypool-j4y1t tetrisv1-5fd57d657d-64m88
mypool-j4y1t tetrisv1-5fd57d657d-65pbj
mypool-j4y1t tetrisv1-5fd57d657d-6gl92
mypool-j4y1t tetrisv1-5fd57d657d-84z9d
mypool-j4y1t tetrisv1-5fd57d657d-9fxcc
mypool-j4y1t tetrisv1-5fd57d657d-cmbgw
mypool-j4y1t tetrisv1-5fd57d657d-fjf4m
mypool-j4y1t tetrisv1-5fd57d657d-gfw2h
mypool-j4y1t tetrisv1-5fd57d657d-ggngx
mypool-j4y1t tetrisv1-5fd57d657d-jscxw
mypool-j4y1t tetrisv1-5fd57d657d-nbhm5
mypool-j4y1t tetrisv1-5fd57d657d-pl856
mypool-j4y1t tetrisv1-5fd57d657d-q79mn
mypool-j4y1t tetrisv1-5fd57d657d-wfm26
mypool-j4y1t tetrisv1-5fd57d657d-xc4sv
mypool-j4y1t tetrisv1-5fd57d657d-xnnjc
mypool-j4y1t tetrisv1-5fd57d657d-z8qnb
mypool-j4y1t tetrisv1-5fd57d657d-zl26f

~/minor-project-2/Tetris-gamev1 git:(main) #3 * do-nyc1-tetrisv1 (1.642s)
kubectl expose deployment tetrisv1 --type=LoadBalancer --port=3000 --target-port=3000

service/tetrisv1 exposed

~/minor-project-2/Tetris-gamev1 git:(main) #3 (2.05s)
doctl compute load-balancer list --format Name,Created,IP,Status

Name Created At IP Status
a117f615b48884f57a2abc780f69d3ad 2024-05-08T06:29:47Z new
```

Fig 3.14

11. ArgoCD

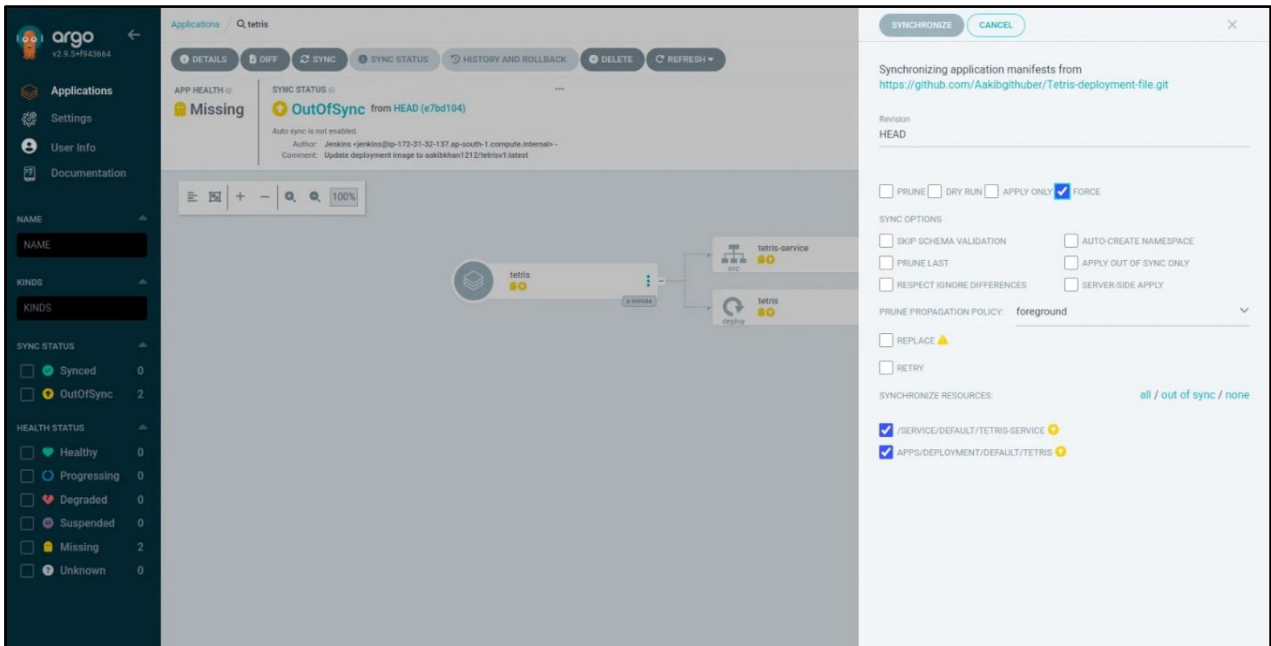


Fig 3.15

12. Kubernetes Dashboard

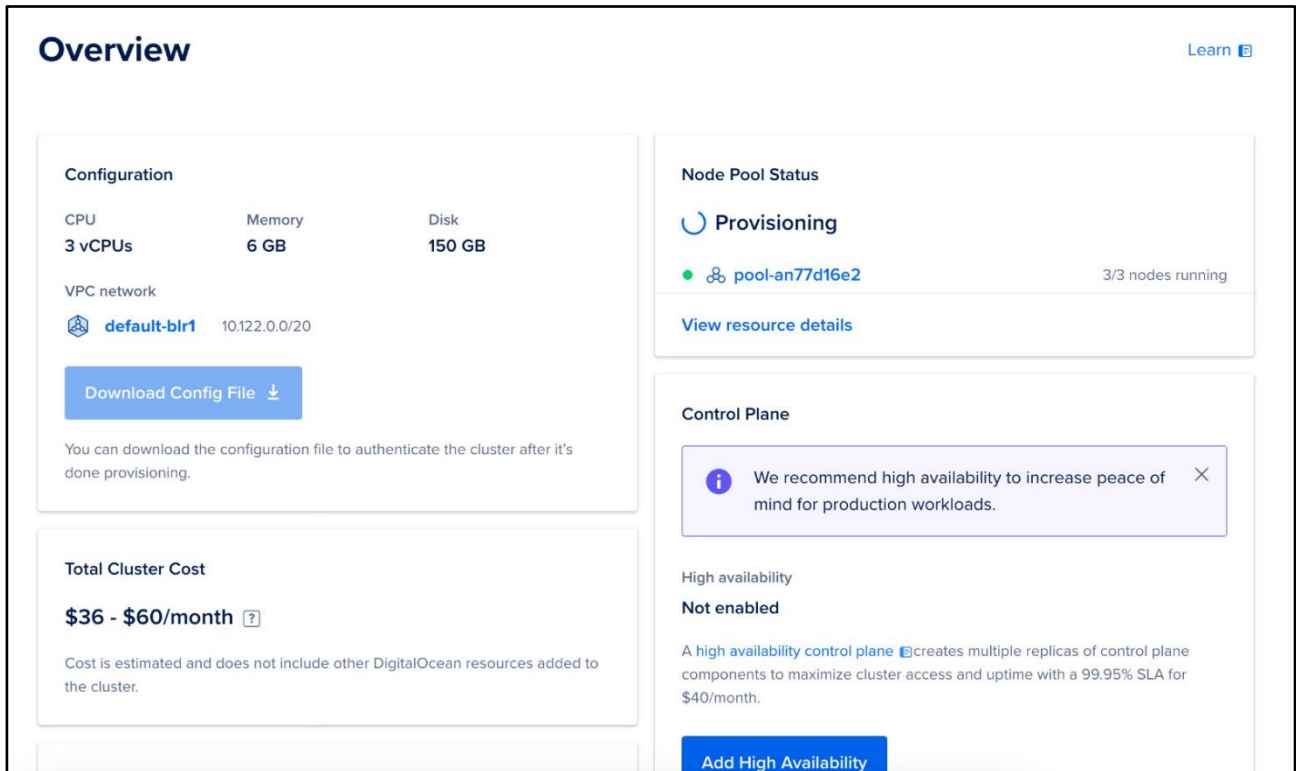


Fig 3.16



Fig 3.17

CHAPTER- 4

MODELLING AND IMPLEMENTATION DETAILS

4.1) DESIGN DIAGRAMS

1. Use Case Diagram

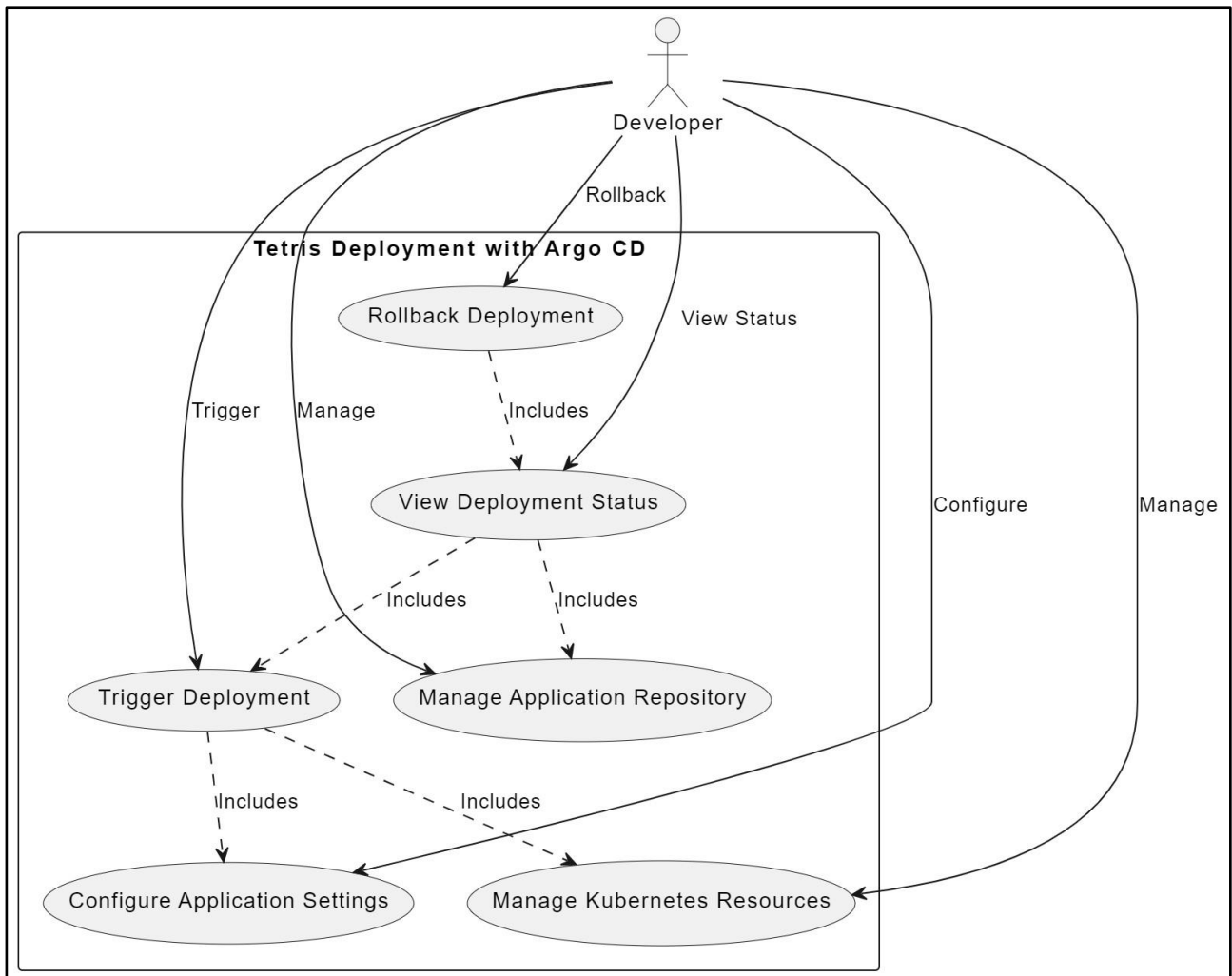


Fig 4.1

2. Application Design Diagram

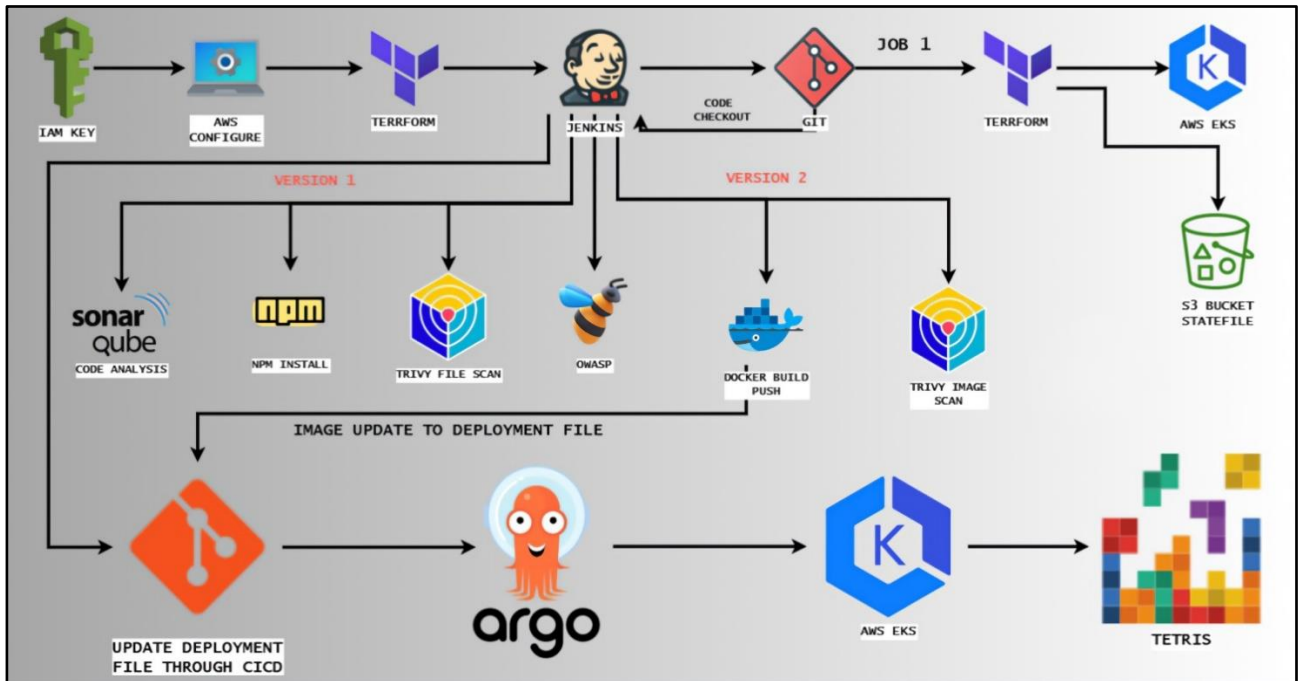


Fig 4.2

3. Control Flow Diagrams

a. Using Kubernetes for deployment processes

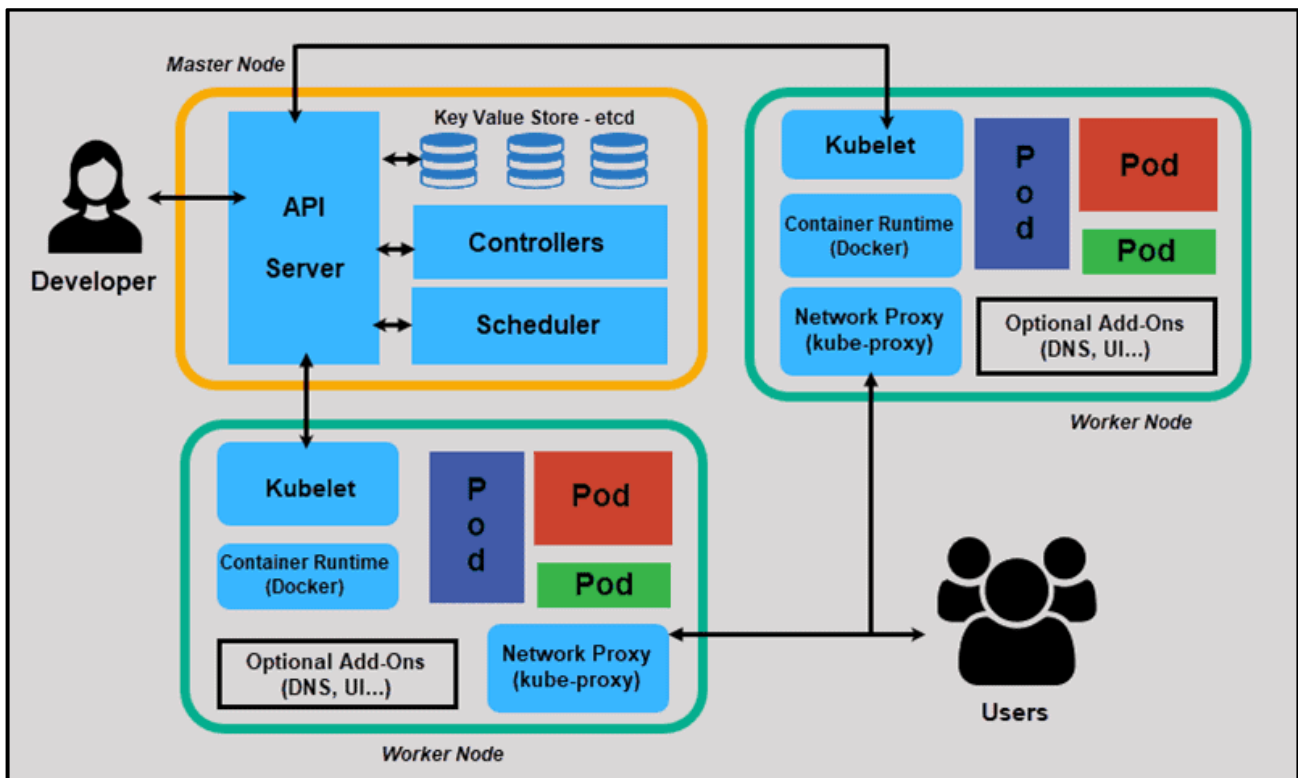


Fig 4.3

b. Trivy for security scanning

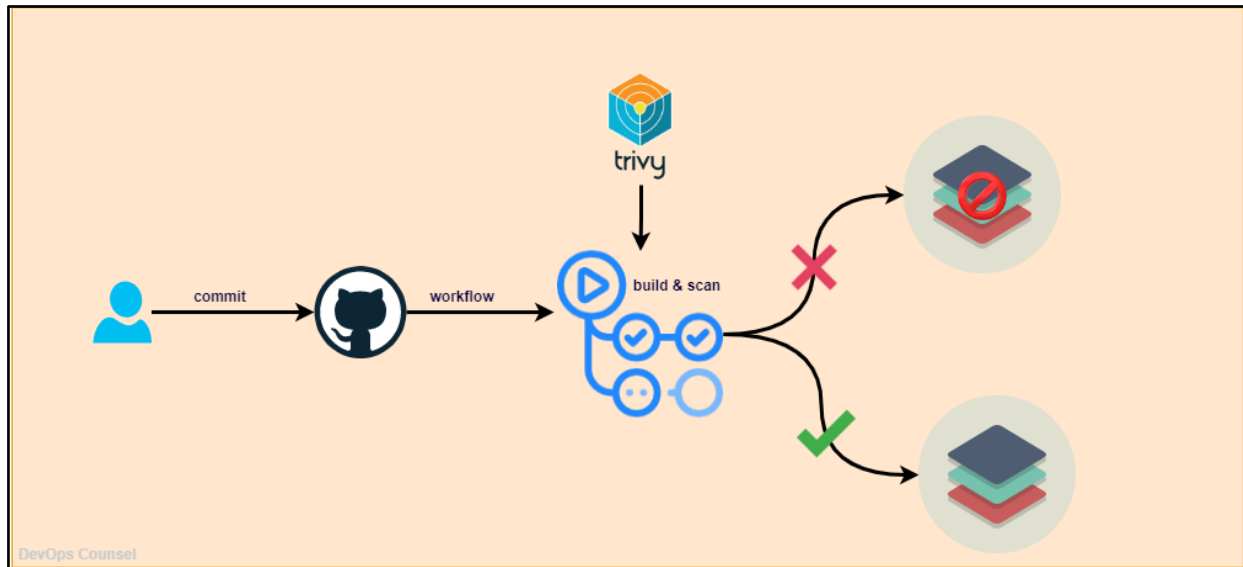


Fig 4.4

4. Activity Diagram

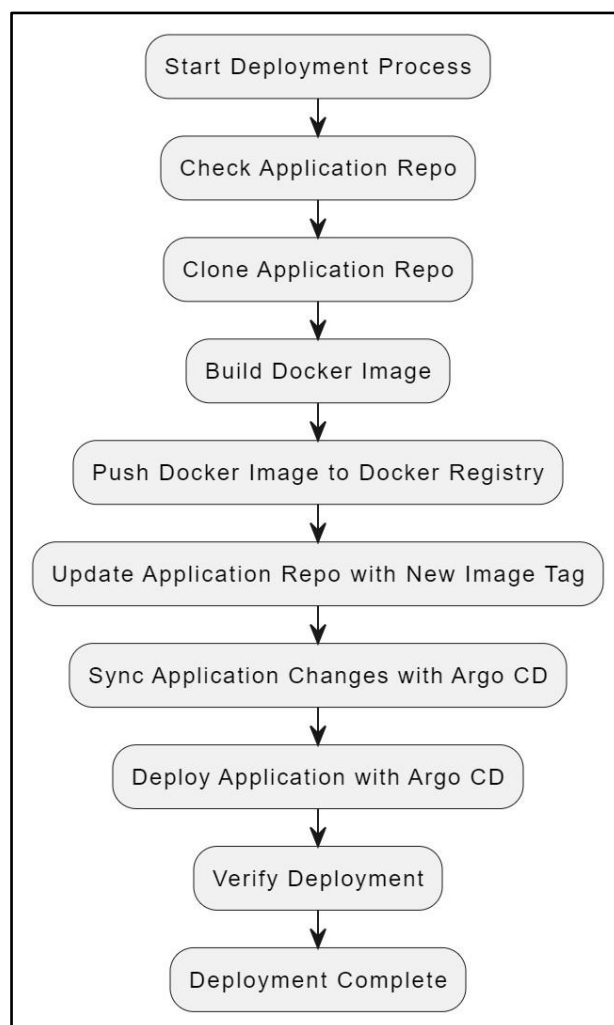


Fig 4.5

4.2) IMPLEMENTATION DETAILS AND ISSUES

+ Implementation Details:

1. Argo CD Installation:

- Definition: Argo CD, a declarative, GitOps continuous delivery tool, was installed on the Kubernetes cluster.

- Technology Role: Argo CD was responsible for managing the deployment lifecycle of Tetris applications, ensuring synchronization with the desired state defined in the Git repositories.

2. DigitalOcean Infrastructure Provisioning:

- Definition: DigitalOcean Droplets were provisioned to host the Kubernetes cluster and associated resources.

- Technology Role: Terraform was used to define and provision the necessary infrastructure components on DigitalOcean, such as Droplets, networking configurations, and storage resources.

3. CI/CD Pipeline Setup:

- Definition: Jenkins pipelines were configured to automate the building, testing, and deployment of Tetris applications.

- Technology Role: Jenkins, integrated with version control systems and Docker, orchestrated the CI/CD process, ensuring rapid and consistent delivery of application updates.

4. Application Deployment with Argo CD:

- Definition: Tetris application manifests were defined and managed within Argo CD, allowing for automated deployment and synchronization with the Git repositories.

- Technology Role: Argo CD facilitated continuous deployment by monitoring changes in the Git repositories and ensuring the desired state of applications in the Kubernetes cluster.

5. Infrastructure Provisioning: Leveraged Terraform to define and provision the required infrastructure on DigitalOcean, including Droplets, networking components, and storage resources.

6. Security Measures: Implemented security best practices, such as role-based access control (RBAC) and network security policies, to protect the deployed applications and infrastructure

+ Issues Encountered:

1. Networking Configuration: Configuring networking settings on DigitalOcean Droplets and

Kubernetes clusters required careful attention to ensure proper communication and access between components.

2. **Argo CD Setup:** Setting up Argo CD and configuring repositories and applications involved a learning curve, especially in understanding YAML manifests and managing application states.

3. **Integration Challenges:** Integrating DigitalOcean resources with Kubernetes and ensuring compatibility between different tools and platforms posed challenges in terms of configuration and troubleshooting.

4. **Resource Management:** Managing resources on DigitalOcean, such as Droplets and storage volumes, required optimization to minimize costs while ensuring adequate performance and scalability.

5. **Automation Complexity:** Automating the deployment pipeline with Argo CD and Jenkins required scripting and workflow design, which introduced complexity in managing and troubleshooting the automation process.

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 Tetris 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 Tetris 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.

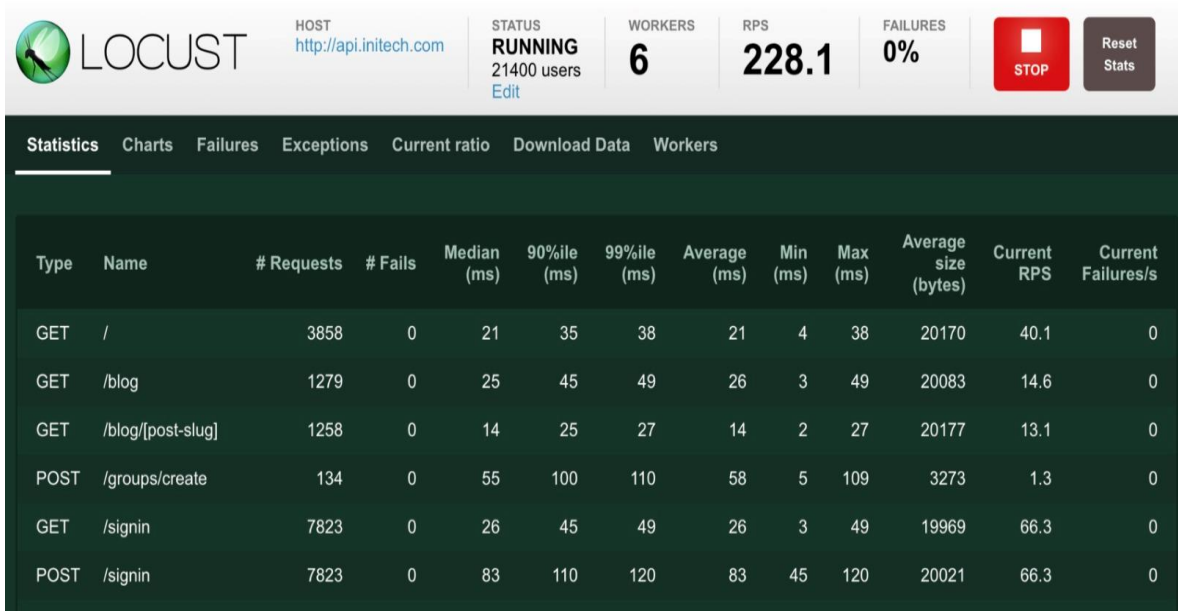


Fig 5.1

The screenshot shows the 'Start new load test' form in the Locust web interface.

Header:

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

Form Fields:

- Number of users (peak concurrency):
- Ramp Up (users started/second):
- Host:
- Advanced options:
- Custom parameters:

Buttons:

- START SWARM** (green)

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 DigitalOcean 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 Tetris 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

1. **Streamlined Deployment Process:** Implementing Argo CD on DigitalOcean significantly streamlined the deployment process by automating the delivery of application updates. This automation reduced manual intervention, minimizing the risk of human error and ensuring consistency across deployments.
2. **Effective Resource Management:** Leveraging DigitalOcean's infrastructure proved highly effective in resource management. The platform's intuitive interface and flexible scalability options allowed for efficient provisioning and management of compute, storage, and networking resources, optimizing resource utilization and cost-efficiency.
3. **Improved Collaboration:** The collaborative efforts of team members in addressing technical challenges and refining deployment processes fostered valuable learning experiences and enhanced team cohesion. Regular communication and knowledge sharing contributed to a shared understanding of project objectives and methodologies.
4. **Enhanced Security Measures:** Integration of security tools such as Trivy and SonarQube, coupled with adherence to security best practices, bolstered the overall security posture of the deployment pipeline. Regular vulnerability scanning and code analysis helped identify and remediate potential security vulnerabilities, ensuring robust protection against threats.
5. **Optimized Performance:** Continuous optimization efforts, including code refactoring, containerization best practices, and infrastructure fine-tuning, led to improved performance and reduced deployment times. Performance monitoring tools provided insights into system behavior, enabling proactive optimization strategies.

6. Reliable Application Management: Argo CD's GitOps-based approach provided reliable application management and version control. Declarative configuration management and automated synchronization ensured consistency between the desired and actual state of applications, facilitating seamless updates and rollbacks.

7. Cost-Efficient Infrastructure: DigitalOcean's transparent pricing model and pay-as-you-go billing offered cost-efficient infrastructure options tailored to project requirements. Rightsizing of resources, strategic utilization of droplets and volumes, and optimization of data transfer costs contributed to cost savings without compromising performance or reliability.

8. Scalability for Future Growth: The deployment pipeline demonstrated scalability, accommodating future growth and increasing workload demands. Scalable infrastructure components, coupled with Kubernetes-based orchestration, provided a foundation for scaling horizontally and vertically as the project evolves.

9. Adaptability to Change: The flexibility inherent in both DigitalOcean's infrastructure and Argo CD's GitOps workflow allowed for adaptability to changing project requirements and environments. Agile methodologies facilitated iterative development cycles, enabling rapid adaptation to evolving business needs and market dynamics.

10. Effective Version Control: Argo CD's integration with version control systems like Git provided effective version management for application configurations and manifests. This enabled the team to track changes, manage rollbacks, and maintain a history of deployments, ensuring consistency and accountability throughout the development lifecycle.

6.2) CONCLUSION

In conclusion, the deployment of the Tetris application using Argo CD on the DigitalOcean infrastructure marks a significant achievement in modern DevOps practices. Through meticulous planning, implementation, and collaboration, we have successfully established a robust deployment pipeline that automates the process of building, testing, and deploying the Tetris application, ensuring efficiency, consistency, and reliability.

By leveraging Argo CD's GitOps methodology, we have streamlined the deployment process, allowing us to declaratively define application configurations and track changes using version control systems. This approach not only simplifies deployment but also enhances visibility, traceability, and auditability of the deployment process, fostering a culture of collaboration and accountability among development teams.

The utilization of DigitalOcean as the cloud infrastructure provider offers scalability, flexibility, and cost-effectiveness, enabling us to provision and manage resources seamlessly while optimizing resource utilization and cost management. The integration of DigitalOcean Droplets and Kubernetes clusters provides a resilient and scalable environment for hosting and orchestrating containerized applications, ensuring high availability and performance.

Throughout the project, we have encountered and addressed various challenges, including infrastructure provisioning, configuration management, and pipeline automation. By overcoming these challenges, we have gained invaluable insights and experience, honing our skills in cloud-native technologies, DevOps practices, and agile methodologies.

Looking ahead, there are ample opportunities for further enhancement and optimization of the deployment pipeline. By embracing continuous improvement, embracing feedback, and staying abreast of emerging technologies and best practices, we can continue to elevate the Tetris application deployment pipeline, delivering greater value, efficiency, and innovation to our stakeholders and users.

In essence, the successful deployment of the Tetris application using Argo CD on DigitalOcean represents not only a technological achievement but also a testament to our dedication, collaboration, and passion for excellence in software development and deployment. As we celebrate this milestone, we remain committed to driving innovation, fostering collaboration, and pushing the boundaries of what is possible in modern software delivery.

6.3) FUTURE WORK

1. **Advanced Monitoring and Logging:** Implement comprehensive monitoring and logging solutions, such as Prometheus and Grafana, to gain insights into application performance, resource utilization, and potential issues within the Kubernetes cluster.
2. **Enhanced Security Measures:** Strengthen security measures by implementing features like Role-Based Access Control (RBAC), network policies, and container security scanning tools to fortify the infrastructure and applications against cyber threats.
3. **Scaling Strategies:** Develop scaling strategies for the Kubernetes cluster to dynamically adjust resource allocation based on workload demands, ensuring optimal performance and cost efficiency.
4. **Fault Tolerance and Disaster Recovery:** Implement fault-tolerant architectures and disaster recovery mechanisms to minimize downtime and data loss in case of infrastructure failures or unforeseen incidents.
5. **GitOps Best Practices:** Refine GitOps workflows and best practices to streamline development, testing, and deployment processes further, ensuring consistency, traceability, and collaboration across development teams.
6. **Continuous Integration Improvements:** Enhance CI/CD pipelines with additional automated tests, code quality checks, and performance benchmarks to maintain high standards of software quality and reliability.
7. **Container Orchestration Optimization:** Explore advanced features of Kubernetes, such as Pod Affinity/Anti-Affinity, Horizontal Pod Autoscaling (HPA), and StatefulSets, to optimize container orchestration and resource utilization.

8. Cost Optimization: Continuously analyze and optimize cloud infrastructure costs by right-sizing resources, leveraging reserved instances, and implementing cost allocation and monitoring tools to ensure efficient resource utilization and budget management.

9. User Experience Enhancements: Solicit feedback from users and stakeholders to identify areas for improving the Tetris application's user interface, performance, and gameplay experience, incorporating new features and enhancements based on user preferences and requirements.

10. Community Engagement and Contributions: Foster a vibrant community around the Tetris project by open-sourcing code repositories, encouraging contributions, and actively engaging with developers, enthusiasts, and contributors to drive innovation and collaboration.

By focusing on these areas of future work, the Tetris project can evolve into a robust, scalable, and feature-rich application, offering an engaging gaming experience while leveraging cutting-edge technologies and best practices in cloud-native development and deployment.

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
9.
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
11. <https://scholar.google.com/>
12. <https://medium.com/>
13. <https://stackoverflow.com/>
14. <https://argoproj.github.io/cd/>