

Cloud-Native CDN Monitoring Using CI/CD

Richa Vivek Savant, S Navin Sunder, Samarth Seshadri, Niharika Panda, Shinu M Rajagopal*

Department of Computer Science and Engineering

Amrita School of Computing, Bengaluru

Amrita Vishwa Vidyapeetham, India

*Corresponding Author: mr_shinu@blr.amrita.edu

Abstract—In a time where digital content consumption reigns, it is essential to efficiently deliver multimedia resources. This research seeks to transform how content is delivered by implementing and overseeing a secure, monitored Content Delivery Network (CDN) on a scalable Kubernetes cluster, managed through a secure CI/CD pipeline built on Jenkins, using the Amazon Web Services (AWS) and essential DevOps tools like Docker, SonarQube, Trivy, Prometheus, Grafana, Argo CD and Helm. The results obtained in this study are derived from an emulation of a CDN. The SonarQube analysis confirms whether the system meets all code quality gates, Trivy identifies and resolves critical container vulnerabilities and OWASP vulnerability checks identify and mitigate security risks in software dependencies. The system is monitored by analyzing the performance metrics visualized in Grafana dashboards, which offer detailed insights into metrics including and not limited to: active jobs, job queue duration, queued rate, memory usage, CPU usage, executor health metrics, system load, RAM usage, root filesystem usage, disk space usage, disk I/O activities, time synchronized drift and time synchronized status. Additionally, SMTP e-mail notifications are configured to improve responsiveness to anomalies identified via monitoring. This study emphasizes making content delivery mechanisms more robust, providing advanced tools for developers, and enabling easy access to digital resources for the community.

Index Terms—Content Delivery Network (CDN), Kubernetes, CI/CD Pipeline, Jenkins, AWS, Docker, SonarQube, Trivy, Prometheus, Grafana, Argo CD, Helm, OWASP, SMTP E-mail Notifications

I. INTRODUCTION

Cloud-Native Applications are created using cloud-native technologies and concepts, with an emphasis on scalability, flexibility, and resilience. Containerization, microservices design, automated scaling, and the application of DevOps techniques are important traits of cloud-native apps. A Content Delivery Network (CDN) is a network of servers spread out to deliver web content to users based on their location, aiming to improve user experience, decrease latency, and speed up content delivery. Balancing traffic loads and offering redundancy improve reliability, ensuring consistent performance even when there are traffic spikes or server failures. CDNs play a vital role in efficiently delivering high-quality video streaming, large downloads, and dynamic content.

Automation is incorporated throughout the software development lifecycle through DevOps techniques, which prioritize collaboration between development and operations teams. This results in enhanced deployment frequency, quicker development cycles, and higher reliability. Modern software develop-

ment incorporates Continuous Integration (CI) and Continuous Deployment (CD) as essential components, ensuring that code changes are automatically tested and deployed to production seamlessly.

In an era dominated by digital content consumption, the efficient delivery of multimedia resources stands as a cornerstone for societal advancement. Security analysis and monitoring can be critical along a CI/CD pipeline in order to ensure that potential threats are identified and mitigated and to spot vulnerabilities, errors, and performance bottlenecks. By integrating monitoring into CI/CD, problems can be found early in the development lifecycle and the release of faulty code cannot be allowed to reach production. This study embarks on a mission to revolutionize the multimedia delivery landscape through the monitoring of a securely deployed Content Delivery Network (CDN) application on a scalable Kubernetes cluster, orchestrated via a secure CI/CD pipeline.

Foundational in the methodology is the seamless integration of the Amazon Web Services with key DevOps tools—Jenkins for pipeline building, Docker for containerization, SonarQube for code quality inspection, Trivy for container security scanning, OWASP for vulnerability assessments, Prometheus and Grafana for monitoring along with SMTP e-mail notifications, Argo CD for continuous delivery on Kubernetes, and Helm for Kubernetes package management—each playing a pivotal role in the deployment process. Through their harmonious collaboration, unparalleled reliability, security, and efficiency can be guaranteed.

This paper proposes the following contributions:

- Integrating SonarQube for code quality inspection, Trivy for container security scanning, and OWASP dependency checks.
- Utilizing Prometheus and Grafana for system monitoring along with effective log analysis and centralized log management.
- Configuring email notifications via SMTP SSL to enhance responsiveness, promptly notifying abnormal behaviour or performance degradation.

This study is divided into multiple sections. Section I which is currently being discussed, provides an introduction regarding the features included in the proposed study. Section II details the literature survey. Section III outlines the proposed methodology. Section IV highlights the deployment results obtained by the application. Section V talks about the advantages observed and future enhancements that can be employed.

II. LITERATURE SURVEY

Abiola et al, in their paper "An Enhanced CI/CD Pipeline: A DevSecOps Approach" discuss about looking at integrating DevSecOps into CI/CD pipelines, with an emphasis on the advantages, problems, and resolutions of improving security as well as efficiency in agile business development. The importance of moving into DevSecOps through Amazon Web Services (AWS) CI/CD tools for securing pipelines and guaranteeing dependable coding will be discussed. However, case studies are not provided in this paper and it lacks empirical data that discuss about practical challenges, which include incorporating security in existing CI/CD pipelines and continuous monitoring for changing security threats. [1]

Nikolov et al, in their paper "Actions Research on the DevSecOps Pipeline" investigate the mutually beneficial connection between machine-driven procedures and human expertise in DevSecOps. It emphasizes timely identification and remediation of vulnerabilities to improve software security. The paper highlights the cultural changes required for effective DevSecOps adoption, focusing on collaboration and agility. It explores tools such as containerization and orchestration for rapid implementation of DevSecOps and acknowledges challenges in threat modeling, secure coding, and regulatory compliance. While secure deployment is critical, it is often seen as a bottleneck to accelerating time to market. [2]

Narendiran et al, in their paper "Integrated log-aware CI/CD pipeline with custom bot for monitoring" propose a self-regulating CI/CD pipeline using GitHub Actions, Terraform, and AWS that minimizes human intervention in software production. Custom server applications process logs and handle errors, which improves error detection, communication, and delivery speed. Simplifying deployment and operations can reduce the cost of training experts, speed up time to market, and improve the entire software development and deployment cycle. [3]

Nandgaonkar et al, in their paper "CI - CD Pipeline for Content Releases" focuses on the Continuous Integration/Continuous Delivery (CI/CD) pipelines with Jenkins and Ansible accelerate software delivery and increase productivity. They enable faster code merges and quality updates, such as content patches. Challenges include automating build tasks, using CI in open source projects, and effective continuous delivery. Adopting a strong CI/CD pipeline can bridge these gaps and enable faster code integration, high-quality releases, and increased software delivery efficiency. [4]

Botez et al, in their paper "Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes" integrate Jenkins and Ansible into CI/CD Pipeline for deploying a Java based web app on AWS via Kubernetes is the main focus of this paper. The importance of DevOps practices, automation and scalability are also stressed upon. It ensures continuous integration is done by Jenkins while Ansible helps in deployment to achieve zero downtime with fast delivery. This shift from traditional IT to DevOps

is hampered by long process within companies and no room for experimentation. The combination of new software tools enables us overcome some CI/CD pipeline implementation risks in terms of automation, security, scalability. [5]

Chaudary et al, in their paper "Cloud DevOps CI -CD Pipeline" focuses on developing a cloud DevOps CI/CD pipeline for microservices with blue/green deployment and highlights the benefits of DevOps in fault detection, integration, and continuous monitoring for secure product delivery. Continuous integration steps include typesetting validation, while continuous deployment involves pushing Docker containers to a custom registry and deploying them to a Kubernetes cluster. Challenges include payment integration and transaction security. It addresses gaps such as lack of DevOps clarity, lack of training, and lack of experienced personnel, and highlights specific challenges in the Chemist Shop development process. [6]

Alawneh et al, in their paper "Expanding DevSecOps Practices and Clarifying the Concepts within Kubernetes Ecosystem" extends DevSecOps practices by integrating security by design principles into the CI/CD lifecycle, focusing on self-managed services that automate infrastructure management and security within the Kubernetes ecosystem. Real-world examples illustrate security integration at each stage and address challenges such as resiliency, availability, and reliability of application delivery. It highlights gaps such as a lack of clarity on security principles, necessary but not discussed practices, and the need for strong security mechanisms in all DevOps processes with a focus on automation and threat awareness for effective risk mitigation. [7]

Kosińska et al, in their paper "Knowledge representation of the state of a cloud-native application" introduce a unified vocabulary for Cloud-native applications and the CNOnt ontology to represent knowledge in this domain. It evaluates the CNOnt Broker's performance in a testbed environment, emphasizing its role in enhancing understanding between microservices and humans. The research underscores the importance of knowledge representation for reasoning on cluster capabilities and mitigating lock-in risks. It addresses gaps by proposing improved understanding of microservices, mitigating lock-in risks, and advocating for further development in ontologies and knowledge management in cloud-native environments. [8]

Chavan et al, in their paper "Implementing DevSecOps Pipeline for an Enterprise Organization" focus on integrating continuous practices such as continuous integration (CI), continuous delivery (CDE), and continuous deployment (CD) into software development. The focus is on automating software creation, testing, and deployment through tools like Jenkins and Selenium. The paper highlights the importance of systematic analysis and synthesis of the continuing practice literature to understand the connections between them. Additionally, it addresses the challenges of automating security testing in DevSecOps pipelines and integrating tools for efficient software process automation. [9]

Yasmine Ska in their paper "A Study and Analysis of Con-

tinuous Delivery, Continuous Integration in Software Development Environment” discuss on the Continuous Integration (CI) and Continuous Delivery (CD) are essential to improving software quality and productivity. CI automates builds and tests, improving code quality and enabling rapid changes. CD ensures robust, automated deployments, eliminating manual tasks and accelerating development. Challenges include maintaining a single code repository, automating processes, testing in a production-like environment, ensuring transparency, managing dependencies, and maintaining different environments and package versions. [10]

Agarwal et al. examine continuous software development with DevOps, emphasizing automated testing tools (e.g., Selenium), Git for version control, Docker for containerization, and Puppet and Chef for configuration management. They advocate user-centered automated acceptance testing to enhance deployment efficiency. Identified challenges include incomplete automated acceptance tests, team communication gaps, manual code reviews despite automation claims, and bureaucratic deployment processes. The study underscores the need for technical and soft skills for effective automation tool use, prioritizing user-goal-based testing to streamline delivery processes. [11]

Rakhi Parashar in their paper “Path to Success with CICD Pipeline Delivery” discuss the importance of continuous integration (CI) and continuous delivery (CD) practices in IT organizations to reduce software delivery time, risk, and cost. It emphasizes benefits such as rapid bug resolution, improved quality assurance, and faster time to market, which impacts the entire software development lifecycle and creates new job roles such as CICD engineers and DevOps engineers. When addressing gaps, focus on defining clear roles and responsibilities, selecting the right tools for each stage, and adapting to the impact of CI/CD on traditional job roles. [12]

Bobrovskis et al, in their paper “A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment” explores the methods, tools, challenges, and practices in continuous integration, delivery, and deployment (CI/CD). Both open source and proprietary solutions are covered, emphasizing the importance of understanding the technology stack for a successful implementation. The document emphasizes careful selection of deployment configuration tools to avoid mistakes. Benefits, challenges, and best practices in automation, programming languages, orchestration tools, and cloud computing are discussed. The lack of a unified solution across different languages and the need for deployment tools with security configuration analysis capabilities are also addressed. [13]

III. METHODOLOGY

In this paper, a secure CI/CD pipeline run by Jenkins is used to build a cloud-native Content Delivery Network (CDN) application on a Kubernetes cluster. The CDN application is encapsulated by Docker containers, which guarantee a standardized runtime environment and easy deployment. These containers are orchestrated by Kubernetes, which also offers load balancing, self-healing, and scalability. Jenkins helps

automate every step of the deployment process, including building, testing, security scanning, and application deployment, from code commit to production.

To ensure quality and security, SonarQube and Trivy are integrated for static code analysis and vulnerability scanning. Prometheus and Grafana monitor application health and performance, providing actionable insights. Argo CD manages continuous deployment, maintaining the desired application state and supporting automated rollbacks. It also seamlessly integrates with Helm, allowing for efficient deployment and management of Helm charts to define, install, and upgrade Kubernetes applications. This comprehensive approach, combining Docker, Kubernetes, Jenkins, SonarQube, Trivy, Prometheus, Grafana, Argo CD and Helm, streamlines the deployment process, ensuring reliability, security, and efficiency. Fig. 1 shows the proposed workflow of the system.

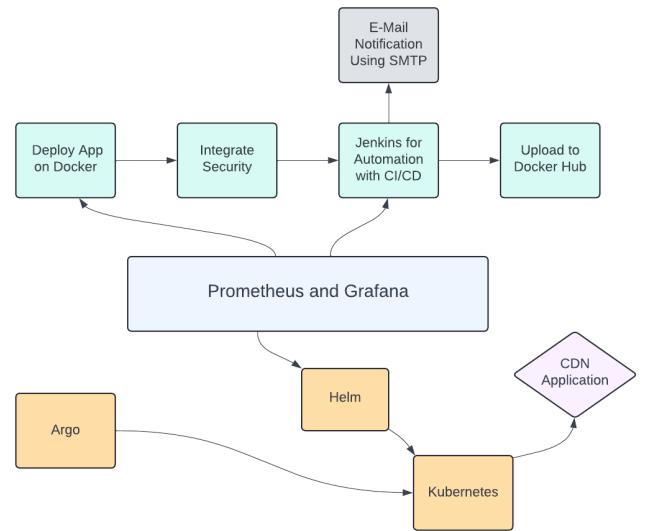


Fig. 1. System Workflow

A. Infrastructure Provisioning and Initial setup

1) *EC2 Instance Setup*: The process begins with launching an EC2 ‘t2.large’ instance for local server deployment. This instance creates an optimal balance of CPU, memory, and network resources. To control traffic to the instance, a new security group is formed and set up to permit only SSH, HTTPS, and HTTP traffic. This makes sure that access to the server is limited to secure and required communication methods. To ensure the proper and secure operation of the web server and other associated services, inbound rules are added to the security group for ports 8081, 8080, and 9000. Additionally, in order to give the program a persistent endpoint and enable dependable internet connection, an Elastic IP is constructed and linked to the EC2 instance.

2) *Application Deployment using Docker Images*: All required files and dependencies are deployed along with the CDN application, and a Docker file with the relevant API

key is generated. Instructions for building the Docker image, including configuring the necessary API key, are included in this Docker file.

The program and its dependencies are packaged as a portable container image using the `docker build` command. Then, using the command `docker run -d -p 8081:80`, the application is launched on the server as a Docker container. In order to make that the application is reachable and capable of processing requests, this command starts the application in a container by mapping port 8081 on the host to port 80 in the container.

B. Building a CI/CD Pipeline with Integrated Security Measures

1) *Incorporating SonarQube, Trivy and OWASP dependency checks for Security:* For SonarQube, a new container image is made. Since SonarQube comes with all the components needed to run it, this container image makes it simple to deploy and maintain it in a Docker environment. The user, after logging into their SonarQube account, can manage projects and adjust security settings. For usage with Jenkins, a token is generated to facilitate automated analysis and two-way connectivity. A manual project is established, and a local analysis is chosen. With the help of this configuration, the user can set up a particular project in SonarQube for code analysis, enabling efficient management and assessment of the code's security and quality.

Next, Trivy is set up. File systems are first examined for security holes. In order to help detect and mitigate any threats, this scanning method looks for any known security flaws or vulnerabilities in the file systems. Subsequently, a targeted scanning of particular docker images is performed, to examine them for weaknesses to make sure that they are free of vulnerabilities that could be used against them in a production setting. Additionally, to mitigate the Log4j vulnerability in installed dependencies, the OWASP dependency check is utilized. This utility makes sure the application is safe and up to date with the most recent security updates by scanning the project's dependencies for known vulnerabilities.

2) *CI/CD Pipeline Automation with Jenkins:* Jenkins is installed followed by the required plugins, which include Docker, Docker Commons, Docker Pipeline, Docker API, Eclipse Temurin Installer, SonarQube Scanner, NodeJS, OWASP Dependency-Check, and docker-buildstep. With the help of these plugins, Jenkins can now perform a variety of functions, including code scanning, managing Node.js applications, and handling Docker operations, such as publishing images to Docker Hub. Jenkins securely authenticates with SonarQube using the credential the SonarQube token is associated with, enabling easy integration and automatic code quality inspection.

The SonarQube and Prometheus servers are to be added with their corresponding URLs under the System settings. Jenkins will be able to communicate with SonarQube for code quality analysis and Prometheus for monitoring and gathering metrics thanks to this setup. JDK, NodeJS, SonarQube

Scanner, Dependency-Check, and Docker are also installed in the tools section, guaranteeing that Jenkins has all the tools required for developing, testing, and deploying applications, offering a complete environment for CI/CD operations.

A Jenkins script is run to build a pipeline as seen in Fig. 2 with stages for workspace cleanup, code checkout, dependency installation, SonarQube analysis, Docker image building, and Kubernetes deployment. OWASP Dependency-Check and Trivy scans are integrated into the pipeline as well to continuously monitor and report on security vulnerabilities during the build process. Docker image building and pushing to DockerHub are automated within the Jenkins pipeline, securely managing DockerHub credentials. Jenkins is used to automate the deployment of Docker containers to the Kubernetes cluster, ensuring seamless application updates.



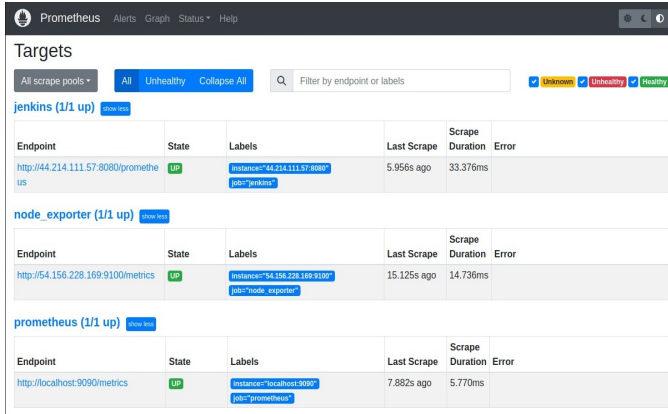
Fig. 2. Building the pipeline on Jenkins

C. Monitoring and Logging Performance Metrics

1) *System Monitoring with Prometheus:* A key-pair is inserted and a 't2.medium' instance is generated. The resources required to operate the monitoring system are provided by this instance, guaranteeing secure access via the key pair for authentication. A security group is created that permits HTTPS and SSH traffic. By limiting access to the instance to secure communication channels, this setup improves security. The security group opens ports 9090 (Prometheus), 9100 (node exporter), and 3000 (Grafana). The Grafana visualization tool, node exporter, and Prometheus monitoring system all depend on these ports to function and be accessible. An elastic IP is assigned to this instance as well.

Prometheus is installed along with promtool for querying. In addition to setting up the main monitoring system, this installation offers management and querying capabilities for the gathered data. To extract the required system metrics in a format that is simple for Prometheus to consume, a node exporter is installed. Prometheus can efficiently gather and process important system metrics thanks to this configuration. A system unit configuration file is created to set up Node exporter and Prometheus as services. The automatic and consistent operation of these services is guaranteed by using a configuration file. Node Exporter is deployed to gather system-level metrics such as CPU usage, memory consumption, and disk I/O from the EC2 instance. Prometheus is configured to scrape metrics from Node Exporter as well as Kubernetes in the future, ensuring comprehensive monitoring coverage.

The Prometheus application, Node Exporter, and Jenkins are all set up to be monitored by the `prometheus.yml` file. This setup can be seen in Fig. 3 and guarantees that Prometheus gathers information from these vital parts, allowing for thorough monitoring. Every instance under observation has a health state of "UP," indicating continuous service availability. The intervals and scrape durations fall within the predicted ranges, guaranteeing prompt data capture for precise monitoring. Hence, the system keeps an eye on the status and performance of both the EC2 instance and the CI/CD pipeline in order to ensure optimal operation.



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
jenkins (1/1 up)					
http://44.214.111.57:8080/prometheus	UP	instance="44.214.111.57:8080" job="jenkins"	5.956s ago	33.376ms	
node_exporter (1/1 up)					
http://54.156.228.169:9100/metrics	UP	instance="54.156.228.169:9100" job="node_exporter"	15.125s ago	14.736ms	
prometheus (1/1 up)					
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	7.882s ago	5.770ms	

Fig. 3. Prometheus Endpoints

2) Visualizing and Analyzing the Metrics with Grafana:

After installation, Grafana is logged into. This installation gives users access to a strong visualization tool that works with Prometheus to provide gathered metrics in an intuitive interface. We choose Prometheus as our data source. Grafana can now access and visualize Prometheus-collected data, enabling more thorough monitoring and analysis.

Dashboards from the Node exporter (id:1860) and Jenkins (id:9964) are imported. These preconfigured dashboards improve the monitoring capabilities and offer useful insights by offering complete views of the performance of Jenkins as well as the node information. The graphs obtained are analyzed in detail in the upcoming section.

3) *Email Notification via SMTP SSL*: To improve overall security and give an extra layer of protection to the email account used for delivering alerts, two-factor authentication (2FA) is enabled. In Jenkins, the SMTP server information and port number (465) are entered, SSL is chosen and the 'Default Trigger' is set to any failures. Lastly, the 'Editable Email Notification' is added to the 'Post-build Actions' section of task settings, the recipients are chosen, and the email content is edited as required.

Jenkins performance and the status of deployment (success or failure) are included in the email messages along with the trivy logs which were created along the pipeline. This configuration ensures that critical information about Jenkins performance and vulnerability assessment results is shared quickly, facilitating timely resolution of any problems.

D. Creating a Kubernetes Cluster

Creating a Kubernetes cluster creates a scalable and elastic environment for deploying and managing containerized applications. It ensures high availability, efficient resource utilization, auto-scaling, and self-healing capabilities, significantly improving application deployment and management.

1) *Creating Appropriate IAM roles*: An IAM role with the `AmazonEKSClusterPolicy` is created before establishing the Amazon EKS cluster. This role is used to create load balancers with Elastic Load Balancing for services by the traditional Cloud Provider and to manage nodes in Kubernetes clusters controlled by Amazon EKS.

The Amazon EKS node kubelet daemon calls AWS APIs and requires that the node have an IAM instance profile and associated policies. Before launching and registering nodes in a cluster, an IAM role is created for the node, applicable to both managed and self-managed nodes using any node AMI. The role requires permissions for the kubelet to describe EC2 resources (`AmazonEKSWorkerNodePolicy`) and to use Amazon ECR container images (`AmazonEC2ContainerRegistryReadOnly` policy). Next, a node group is created with the IAM role. A 't3.medium' instance type is selected, and the maximum node count is set to 1. Additionally, inbound rules are added for port 30007 for the application node and port 9100 for the node exporter.

2) *Integrating Argo CD and Helm for Monitoring*: The `kubeconfig` file is generated for the Amazon EKS cluster to facilitate the management of the Kubernetes cluster from a local machine. Argo CD components are installed in the Kubernetes cluster and the Argo CD CLI is set up on the local machine. Similarly, Helm is installed.

To begin monitoring the Kubernetes cluster, the Prometheus Node Exporter is installed to collect system-level metrics from the cluster nodes. A Kubernetes namespace for the Node Exporter is created, and the Node Exporter is installed using Helm. A load balancer is employed to expose the 'argocd-server'. The server's hostname is obtained when the load balancer is established, and the initial password is autogenerated with the pod name of the Argo CD API server. This password is used to log in. The CDN application repository is then connected using HTTPS within Argo CD and a new app is created which will then fetch the Kubernetes Manifest file. The node port service is used at port 30007 on which the CDN application successfully runs.

Next, a job is added to the 'prometheus.yml' file to scrape metrics from 'nodeip:9001/metrics', by updating the configuration with a new job, specifying the job name and target IP. Hence, a scalable Kubernetes cluster with node groups is established, monitored using Prometheus and Node Exporter, and application deployment is managed with Argo CD.

IV. RESULTS AND ANALYSIS

Using the aforementioned security and monitoring tools, the deployed pipeline produced important insights and outcomes. A thorough examination of the results is carried out.

1) *Results of SonarQube, Trivy and OWASP dependency checks:* The output of the SonarQube code analysis as seen in Fig. 4 shows that the project successfully completed all quality gate requirements. The code is free of errors and weaknesses. Four security hotspots are located. Since there aren't many reviewed security hotspots, the security review is rated 'E,' however the code's dependability and security are both rated 'A'. Additionally, the code maintainability is rated 'A,' with 18 code smells found and a technical debt of one hour and forty-three minutes.

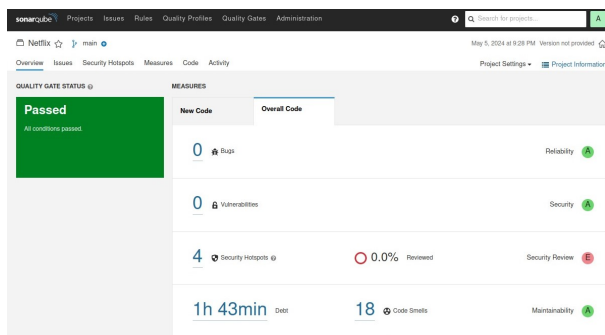


Fig. 4. SonarQube Code Analysis

The Trivy file system vulnerability assessment as seen in Fig. 5 shows that a critical vulnerability was found in the 'yarn.lock' file that is associated with the 'json5' library. This vulnerability 'CVE-2022-46175', is related to prototype pollution in 'json5' through the parse method. The 'json5' version 2.2.1 has a vulnerability that is patched in versions 2.2.2 and 1.0.2. Thus, the vulnerability has been addressed.

```
yarn.lock (yarn)
```

Total: 1 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
json5	CVE-2022-46175	HIGH	fixed	2.2.1	2.2.2, 1.0.2	json5: Prototype Pollution in JSON5 via Parse Method https://www.exploit-db.com/exploits/46175/

Fig. 5. Trivy File System Vulnerability Analysis

The CI/CD pipeline orchestrated through Jenkins demonstrates seamless automation from code integration to deployment. The pipeline stages including SonarQube analysis, OWASP dependency checking, and Docker operations are all executed efficiently. A minor deployment error was discovered and resolved, highlighting the importance of continuous monitoring and rapid response mechanisms.

2) *Analysis of the Generated Grafana Dashboards:* The integration of Prometheus and Grafana enables the comprehensive visualization of the system metrics. The Grafana dashboards customized specifically for Jenkins and Node Exporter provide intuitive insights into the real-time system performance and health. It is observed that the key performance indicators remain within optimal ranges, confirming the stability and reliability of the monitored infrastructure.

The Jenkins dashboard of Grafana, as seen in Fig. 6 gives a detailed summary of Jenkins' performance and health metrics.

The processing speed, job queue duration, and queued rate are all at zero, which means that there are currently no jobs being processed or waiting in the queue. Memory usage is reported as 1GB, and Jenkins health is measured at 1.0. The duration of JVM operation cannot be determined as 'N/A,' indicating a lack of data. The CPU is using very little at only 0.00495 percent. All Jenkins nodes are online and there are 2 free executors, showing that there are available executors. There has been little recent activity shown in the graphs depicting the duration of the job and the health of the executor.

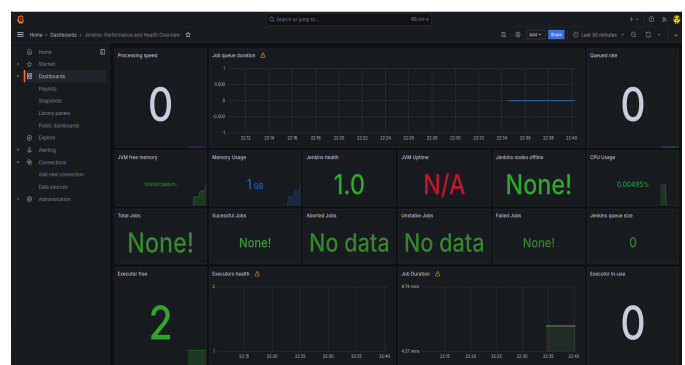


Fig. 6. Jenkins Performance Monitoring in Grafana

The Node Exporter dashboard of Grafana presents a thorough overview of various monitored components, including:

- Quick CPU / Mem / Disk
- Basic CPU / Mem / Net / Disk
- CPU / Memory / Net / Disk
- Memory Meminfo
- Memory Vmstat
- System Timesync
- System Processes
- Systemd
- Storage Disk
- Storage Filesystem
- Network Traffic
- Network Sockstat
- Network Netstat

The 'Quick CPU / Mem / Disk' and 'Basic CPU / Mem / Net / Disk' component measurements can be seen in Fig. 7. The section labeled 'Quick CPU / Mem / Disk' displays the system's stress indicators, such as CPU usage at 0.2 percent. The CPU utilization metric stands at 0.7 percent, with system load at 3.0 percent, RAM usage at 13.3 percent, and root filesystem usage at 15.5 percent. The host has 2 CPU cores and has been up for a total of 3.6 minutes. The section 'Basic CPU / Mem / Net / Disk' displays graphs of CPU usage over a period of time, depicting a system that is mostly inactive with minimal CPU utilization. There is low network traffic, and the basic memory usage shows a total of 4 GiB of RAM. There are no substantial changes shown in the visualization of disk space usage. In general, the dashboard shows that the system is stable with low load.

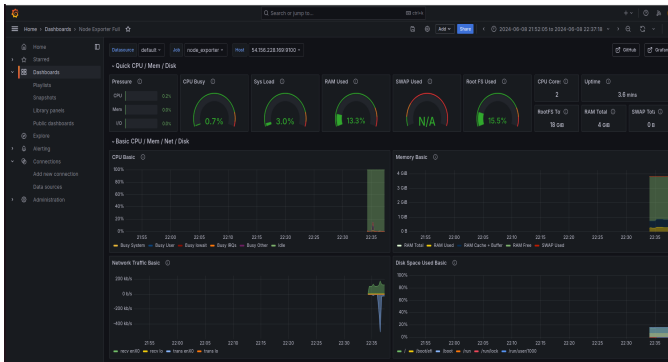


Fig. 7. Node Exporter Performance Monitoring in Grafana - 'Quick CPU / Mem / Disk' and 'Basic CPU / Mem / Net / Disk'

The 'CPU / Memory / Net / Disk' component measurements as shown in Fig. 8 displays in-depth statistics about network traffic, disk space usage, disk I/O operations, and I/O usage. The network traffic data displays the mean, latest, and highest data amounts received and sent showing slight changes and a generally low rate of data transfer. Display of disk space usage is shown for different filesystems, with notable usage observed in the '/boot/efi' and '/boot' partitions. The system monitors Disk I/O activities on the 'xvda' disk, recording both read and write operations with sporadic spikes in write activities despite overall low movement. The graph of I/O usage shows the amount of bytes read and written over time, indicating minimal read and write operations. To generalize, the dashboard shows a stable and low-load system environment.

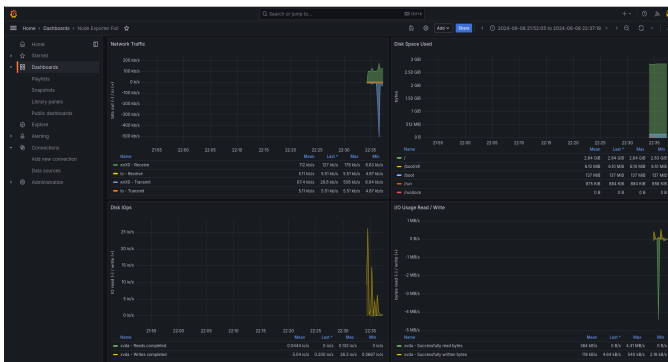


Fig. 8. Node Exporter Performance Monitoring in Grafana - 'CPU / Memory / Net / Disk'

The 'System Timesync' component measurements as shown in Fig. 9 offers comprehensive metrics on system time synchronization. The panel for "Time Synchronized Drift" displays the estimated error in seconds, with an average of 5.85 microseconds and a most recent value of 3 microseconds. The "Time PLL Adjust" panel shows 2 consistent phase-locked loop adjustments during the monitored period. The panel for "Time Synchronized Status" indicates that the system clock is in sync with a dependable server, with an average local clock frequency adjustment of 1.0. The panel named "Time Misc"

consistently shows a 10-millisecond gap between clock ticks, along with a fixed offset of 37 seconds from International Atomic Time (TAI). In general, the dashboard shows that the monitored system has accurate and stable time synchronization.

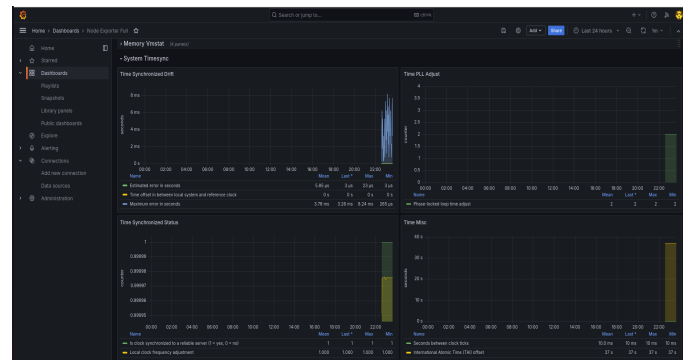


Fig. 9. Node Exporter Performance Monitoring in Grafana - 'System Timesync'

Hence, the monitoring tools successfully provide real-time monitoring and visualization to help proactively resolve issues. To maximize system performance and dependability, a variety of measures can be implemented in response to the observed metrics as obtained by these tools. The metrics like CPU utilization, memory consumption, and disk I/O can be examined to identify bottlenecks or inefficiencies in the system. Configurations can then be adjusted, resources allocated, and if necessary, scaled to ensure optimal performance and mitigate potential issues effectively. Proactive capacity planning is made possible by historical data from disk space consumption and network traffic indicators. This allows for upgrades or expansions to be made on time to accommodate future demands. Hence, monitoring insights can be used to optimize deployment procedures and CI/CD pipelines by putting improvements in place to speed up workflows, lower mistake rates, and boost overall effectiveness.

Additionally, it is observed that the email notifications successfully provide timely updates on Jenkins performance and deployment results along with Trivy vulnerability analysis reports. This proactive alerting mechanism ensures that stakeholders are immediately informed of all critical events and can intervene quickly if necessary.

3) *Enhanced Deployment Scalability with Argo CD and Helm in Kubernetes:* Utilizing Argo CD and Helm with Kubernetes for deploying the application highlights the scalability and flexibility of the selected orchestration tool. Argo CD, a GitOps continuous delivery tool for Kubernetes that is declarative, guarantees smooth and automated deployments of applications. Helm makes it easier to manage Kubernetes applications by providing a package manager that allows for simpler updates and rollbacks. Kubernetes coordinates the deployment, scaling, and operation of containerized applications to guarantee high availability and resilience. This comprehensive method improves the deployment of applications, boosts

operational efficiency, and offers strong systems for managing failures and adjusting workloads dynamically, creating a resilient and highly accessible application environment.

V. CONCLUSION

In conclusion, the integration of Jenkins, Docker Hub, SonarQube, Trivy, Prometheus, Grafana, and Kubernetes significantly improves the overall performance of Content Delivery Network (CDN) applications running in cloud environments. These tools simplify the deployment process, ensure secure delivery and provide monitoring capabilities, resulting in a robust, efficient, and scalable application infrastructure.

The research describes a strong CI/CD pipeline that leverages contemporary cloud-native technologies to efficiently and securely deploy a CDN application. Through incorporating automation throughout the entire process - from code submission to deployment - it guarantees uniform environments and thorough testing, ultimately improving dependability and expandability. The utilization of Docker and Kubernetes technologies enables smooth deployment and control of applications, improving resource usage and allowing quick scalability. The security measures concentrate on vulnerability scanning and code analysis, and are integrated into the pipeline to uphold the integrity of applications. Monitoring tools offer immediate visibility into application performance and health, enabling proactive management and quick problem resolution.

This study is an initial move towards creating a complete system tailored for practical CDN uses. Although the current results are obtained from emulating a CDN application, the knowledge obtained from measuring metrics like CPU usage, memory usage, and disk I/O is extremely valuable. The data from the monitoring tools show the possibility of pinpointing and fixing inefficiencies in live deployment settings. The next step will involve implementing this framework in operational CDN systems, where live data can improve deployment, resource management, and performance. This shift from simulated to actual settings will allow the system to tackle practical obstacles, ultimately leading to stronger, more effective, and easily expandable CDN application infrastructures. The proactive capacity planning and optimization strategies discussed can be put into practice, resulting in significant enhancements in workflow efficiency, error reduction, and overall system effectiveness. Additionally, leveraging AWS Lambda@Edge for edge computing can be examined to boost response times and security in the future. Chaos Engineering can also be implemented using tools like Gremlin or Chaos Monkey to test system resilience by simulating failures. AI/ML tools can be leveraged as well, for predictive analytics and proactive issue detection.

REFERENCES

- [1] O. B. Abiola and O. G. Olufemi. (2022). "An Enhanced CICD Pipeline: A DevSecOps Approach." *International Journal of Computer Applications*, 975, p. 8887. Available: *International Journal of Computer Applications*.
- [2] L. A. Nikolov and A. P. Aleksieva-Petrova. (2023, September). "Action Research on the DevSecOps Pipeline." In *2023 International Scientific Conference on Computer Science (COMSCI)* [Conference paper], pp. 1-6. Available: *IEEE Xplore*.
- [3] A. Narendiran, D. Abhishek, P. Adithya, D. Ray, P. K. Auradkar, and H. L. Phalachandra. (2023, April). "Integrated log-aware CI/CD pipeline with custom bot for monitoring." In *2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)* [Conference paper], pp. 257-262. Available: *IEEE Xplore*.
- [4] S. Nandgaonkar and V. Khatavkar. (2022, October). "CI-CD pipeline for content releases." In *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)* [Conference paper], pp. 1-4. Available: *IEEE Xplore*.
- [5] A. Cepuc, R. Botez, O. Craciun, I. A. Ivanciu, and V. Dobrota. (2020, December). "Implementation of a continuous integration and deployment pipeline for containerized applications in Amazon Web Services using Jenkins, Ansible and Kubernetes." In *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)* [Conference paper], pp. 1-6. Available: *IEEE Xplore*.
- [6] G. Bou Ghantous and A. Gill. (2017). "DevOps: Concepts, practices, tools, benefits and challenges." *PACIS2017*. Available: *ResearchGate*.
- [7] M. Alawneh and I. M. Abbadi. (2022). "Expanding DevSecOps Practices and Clarifying the Concepts within Kubernetes Ecosystem." In *2022 Ninth International Conference on Software Defined Systems (SDS)* [Conference paper], pp. 1-7. Available: *IEEE Xplore*, doi: 10.1109/SDS57574.2022.10062874.
- [8] J. Kosińska, G. Brotoń, and M. Tobiasz. (2024). "Knowledge representation of the state of a cloud-native application." *International Journal on Software Tools for Technology Transfer*, vol. 26, no. 1, pp. 21-32. Available: *Springer*.
- [9] S. V. Deshmukh, D. S. Ahire, N. N. Chavan, N. D. Bharambe, and A. R. Jain. "Implementing DevSecOps pipeline for an enterprise organization." Available: *ResearchGate*.
- [10] Y. Ska and P. Janani. (2019). "A study and analysis of continuous delivery, continuous integration in software development environment." *International Journal of Emerging Technologies and Innovative Research*, vol. 6, pp. 96-107. Available: *JETIR*.
- [11] A. Agarwal, S. Gupta, and T. Choudhury. (2018, June). "Continuous and integrated software development using DevOps." In *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)* [Conference paper], pp. 290-293. Available: *IEEE Xplore*.
- [12] R. Parashar. (2021). "Path to success with CI/CD pipeline delivery." *International Journal of Research in Engineering, Science and Management*, vol. 4, no. 6, pp. 271-273. Available: *IJRESM*.
- [13] S. Bobrovskis and A. Jurenoks. (2018). "A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment." In *BIR Workshops* [Conference paper], pp. 314-322. Available: *CEUR-WS*.
- [14] K. D. Kumar and E. Umamaheswari. (2017). "An authenticated, secure virtualization management system in cloud computing." *Asian Journal of Pharmaceutical and Clinical Research*. Available: *AJPCR*.
- [15] K. D. Kumar and E. Umamaheswari. (2018). "Resource provisioning in cloud computing using prediction models: A survey." *International Journal of Pure and Applied Mathematics*, vol. 119, no. 9, pp. 333-342. Available: *IJPAM*.
- [16] B. Karthikeyan, T. Sasikala, and S. B. Priya. (2019). "Key exchange techniques based on secured energy efficiency in mobile cloud computing." *Applied Mathematics Information Sciences*, vol. 13, no. 6, pp. 1039-1045. Available: *Natural Sciences Publishing*.
- [17] T. Bikku. (2023). "Fuzzy associated trust-based data security in cloud computing by mining user behavior." *International Journal of Advanced Intelligence Paradigms*, vol. 25, no. 3-4, pp. 382-397. Available: *Inder-science*.
- [18] M. R. Reddy, R. Akilandeswari, S. Priyadarshini, B. Karthikeyan, and E. Ponmani. (2017, July). "A modified cryptographic approach for securing distributed data storage in cloud computing." In *2017 International Conference on Networks Advances in Computational Technologies (NetACT)* [Conference paper], pp. 131-139. Available: *IEEE Xplore*.
- [19] D. Prabhu, S. V. Bhanu, and S. Suthir. (2022). "Privacy preserving steganography based biometric authentication system for cloud computing environment." *Measurement: Sensors*, vol. 24, p. 100511. Available: *Elsevier*.