**NETFLIX CLONE APP USING DEVSECOPS PIPELINE**

**1. Executive Summary**

As organizations transition to **cloud-native architectures**, DevSecOps plays a critical role in accelerating software delivery, improving system reliability, and enhancing security. This project implements an **automated CI/CD pipeline** that ensures **seamless integration, security validation, containerization, deployment, and monitoring**. The solution is designed to meet **enterprise-grade reliability, security, and scalability requirements** while aligning with **industry best practices**.

**Key Benefits:**

* Automated, Secure, and Scalable Deployments
* Enhanced Code Quality & Security Compliance
* Efficient Resource Utilization with Kubernetes & Docker
* Comprehensive Observability with Prometheus & Grafana

**2. Business Objectives & Challenges Addressed**

**Business Objectives:**

1. **Accelerate Software Delivery** – Reduce deployment time through automation.
2. **Enhance Security & Compliance** – Implement shift-left security by integrating vulnerability scans.
3. **Improve System Resilience** – Deploy fault-tolerant workloads on Kubernetes.
4. **Minimize Operational Overhead** – Automate build, testing, and deployment workflows.

**Challenges Addressed:**

| **Challenges** | **Solution Implemented** |
| --- | --- |
| **Manual, error-prone deployments** | Automated CI/CD pipeline with Jenkins |
| **Security vulnerabilities in code & dependencies** | Static & dynamic security scanning (SonarQube, OWASP ZAP, Trivy) |
| **Scalability & infrastructure management** | Containerization (Docker) & orchestration (Kubernetes) |
| **Limited visibility into system health** | Real-time monitoring (Prometheus, Grafana) |

**3. Technical Architecture Overview**

The pipeline follows a **multi-stage DevSecOps lifecycle**, ensuring continuous integration, security validation, and deployment automation.

**3.1 Architecture Diagram**

*A screen shot of a diagram

AI-generated content may be incorrect.*

**3.2 Key Components & Workflow**

**Step 1: Source Code Management & Version Control**

* Developers commit code to **GitHub**, triggering an automated build in **Jenkins**.
* **Branching strategy (e.g., GitFlow)** is followed for structured development and releases.

**Step 2: Code Quality & Security Validation**

* **SonarQube Analysis** – Performs **static code analysis**, detecting code smells, bugs, and vulnerabilities.
* **NPM Dependency Check** – Identifies **outdated or vulnerable dependencies** within the project.
* **Trivy File Scan** – Scans source code for **secrets, misconfigurations, and vulnerabilities**.

**Step 3: Security & Compliance Enforcement**

* **OWASP ZAP (DAST)** – Performs **penetration testing** to uncover security loopholes in the application.

**Step 4: Containerization & Image Security**

* **Docker Build & Push** – The application is containerized and stored in a **private container registry**.
* **Trivy Image Scan** – Ensures container images meet security and compliance requirements.

**Step 5: Deployment Strategy**

* **Deploy on Container** – Runs the application in a **containerized environment** for consistency.
* **Deploy to Kubernetes** – Uses **Kubernetes orchestration** to enable **scalability, high availability, and load balancing**.
* **Deployment Strategies Implemented:**
  + **Rolling Updates** – Gradual deployment to minimize downtime.
  + **Blue-Green Deployments** – Ensuring zero downtime by running two environments in parallel.

**Step 6: Monitoring, Logging & Alerts**

* **Prometheus & Grafana** – Provides **real-time performance monitoring and dashboards**.
* **Alerting Mechanisms** – Sends alerts based on performance degradation or failures.
* **Email Notifications** – Jenkins notifies stakeholders of pipeline execution status.

**PHASE – 1**

**TOOLS SETUP**

The first phase of implementing the Netflix clone CI/CD pipeline involves setting up the essential DevOps tools required for **automation, security, containerization, and monitoring**. This step lays the foundation for a fully automated software delivery process, ensuring **scalability, reliability, and security** from the start.

**A screenshot of a computer

Description automatically generated**

* + 1. **Jenkins**

**A screenshot of a computer

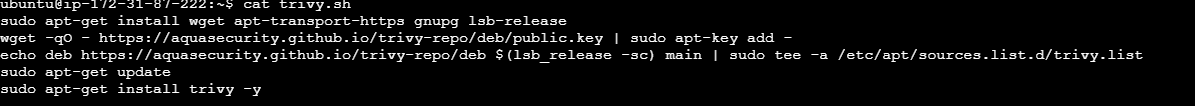
Description automatically generated**

* + 1. **SonarQube**

**A screenshot of a computer

Description automatically generated**

* + 1. **Trivy Scanner**

****

**A screenshot of a computer program

Description automatically generated**

* + 1. **Create a TMBD API key**

A **Netflix clone app** requires an extensive catalog of movies and TV shows, along with metadata like:  
 **Movie titles, descriptions, and genres**  
 **Posters, backdrops, and trailers**  
 **Ratings, reviews, and trending content**  
 **Actor and director details**

Instead of manually storing and updating movie data, we **use the TMDB API** to dynamically fetch the latest movies, trending shows, and search results.

To access TMDB’s movie database programmatically, you need an API Key, which serves as an authentication mechanism. The API key allows your application to:

🔹 Send requests to TMDB’s servers

🔹 Retrieve movie and TV show data

🔹 Prevent unauthorized access and limit API rate usage

**A screenshot of a computer

Description automatically generated**

* + 1. **Install Prometheus and Grafana**

First of all, let’s create a dedicated Linux user sometimes called a system account for Prometheus. Having individual users for each service serves two main purposes:

It is a security measure to reduce the impact in case of an incident with the service.

It simplifies administration as it becomes easier to track down what resources belong to which service.

A screenshot of a computer

Description automatically generated

* + 1. **Install Node Exporter on Ubuntu 22.04**

Next, we’re going to set up and configure Node Exporter to collect Linux system metrics like CPU load and disk I/O. Node Exporter will expose these as Prometheus-style metrics.

A screenshot of a computer

Description automatically generated

To create a static target, you need to add job\_name with static\_configs in Prometheus.yaml config file.

A screenshot of a computer program

Description automatically generated

In the promethues dashboard, under target section our node exporter is successfully installed.

A screenshot of a computer

Description automatically generated

To visualize metrics we can use Grafana. There are many different data sources that Grafana supports, one of them is Prometheus.

First, let’s make sure that all the dependencies are installed.

A screenshot of a computer

Description automatically generated

Now integrate the Grafana with Prometheus by setting up a dashboard and selecting the source as Prometheus.

A screenshot of a computer program

Description automatically generated

Click on Import Dashboard paste this code 1860 and click on load. Select the Datasource and click on Import.

A screenshot of a computer

Description automatically generated

**PHASE – 2**

**INTEGRATION BETWEEN THE TOOLS**

* 1. **Integrate Prometheus and Grafana with Jenkins**

A screenshot of a computer

Description automatically generated

To create a static target, you need to add job\_name with static\_configs. go to Prometheus server

A computer screen shot of a program

Description automatically generated

As we can see in the Prometheus targets, Jenkins is successfully integrated.

A screenshot of a computer

Description automatically generated

Let’s add Dashboard for a better view in Grafana

Click On Dashboard –> + symbol –> Import Dashboard

Use Id 9964 and click on load. We can monitor Jenkins metrics here.

A screenshot of a computer

Description automatically generated

* 1. **Email Integration With Jenkins and Plugin Setup**

Install Email Extension Plugin in Jenkins

A screenshot of a computer

Description automatically generated

Go to your Gmail and click on your profile

Then click on Manage Your Google Account –> click on the security tab on the left side panel you will get this page(provide mail password).

2-step verification should be enabled.

Search for the app in the search bar you will get app passwords like the below image

Once the plugin is installed in Jenkins, click on manage Jenkins –> configure system there under the E-mail Notification section.

A screenshot of a computer

Description automatically generated

Now under the Extended E-mail Notification section configure the details as shown in the below images.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Click on Apply and save.

**3. Install Plugins like JDK, Sonarqube Scanner, NodeJs, OWASP Dependency Check**

Goto Manage Jenkins → Tools → Install JDK(17) and NodeJs(16)→ Click on Apply and Save.

A screenshot of a computer

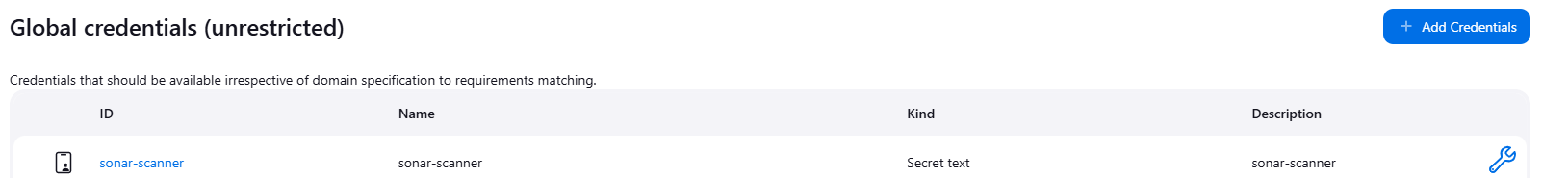
Description automatically generated

SONAR PLUGIN

Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, so <Public IP>:9000. Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token

copy Token

Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text.



Now, go to Dashboard → Manage Jenkins → System and Add like the below image.

A screenshot of a computer

Description automatically generated

**The Configure System option** is used in Jenkins to configure different server

**Global Tool Configuration** is used to configure different tools that we install using Plugins

We will install a sonar scanner in the tools.

A close-up of a login

Description automatically generated

Create a webhook configuration in sonarqube and give Jenkins URL as the destination.

A screenshot of a computer

Description automatically generated

OWASP PLUGIN

GotoDashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install it without restart.

A screenshot of a computer

Description automatically generated

First, we configured the Plugin and next, we had to configure the Tool

Goto Dashboard → Manage Jenkins → Tools

A blue lines on a white background

Description automatically generated

DOCKER PLUGIN

We need to install the Docker tool in our system, Goto Dashboard → Manage Plugins → Available plugins → Search for Docker and install these plugins

Docker

Docker Commons

Docker Pipeline

Docker API

docker-build-step

A screenshot of a computer

Description automatically generated

Now, goto Dashboard → Manage Jenkins → Tools →

A screenshot of a computer

Description automatically generated

In the Jenkins global credentials, store the docker username and password.

A screenshot of a computer

Description automatically generated

WRITING DOCKERFILE

A screen shot of a computer

Description automatically generated

This **Dockerfile** defines a two-stage build process for a **Netflix clone app** using Node.js, Yarn, and Nginx. In the first stage (**builder**), it uses the **Node.js 16.17.0-alpine** base image, sets up the working directory, installs dependencies from package.json and yarn.lock, and builds the application while injecting the **TMDB API key** (TMDB\_V3\_API\_KEY) as an environment variable for API access. The second stage uses an **Nginx stable-alpine** image, clears any existing files, and copies the built application (/app/dist) from the builder stage to Nginx's default web directory. It then **exposes port 80** and sets **Nginx as the entry point**, ensuring the app is served efficiently as a static web application.

**PHASE – 3**

**CREATE A PIPELINE JOB IN JENKINS**

Create a pipeline job

A screenshot of a computer

Description automatically generated

Let us write the pipeline script till the docker stage and build the pipeline.

A computer screen shot of text

Description automatically generated

**Pipeline Explanation (Stage-wise)**

**1. Clean Workspace**

This stage ensures a fresh build environment by removing any leftover files from previous builds. It prevents conflicts caused by cached or outdated files.

**2. Checkout from Git**

The latest code from the GitHub repository (main branch) is fetched and prepared for the CI/CD pipeline execution.

**3. SonarQube Analysis**

This stage performs **static code analysis** using **SonarQube**, checking for bugs, code smells, and security vulnerabilities. It assigns a **SonarQube project name and key** to track quality reports.

**4. Quality Gate**

This stage verifies if the **SonarQube Quality Gate** passes or fails. If issues are found, the pipeline continues execution but logs the results for developers to review.

**5. Install Dependencies**

Using **npm install**, the required **Node.js** dependencies are installed, ensuring that the application has all the necessary libraries and modules.

A screenshot of a computer program

Description automatically generated

**6. OWASP Dependency Check**

The **OWASP Dependency Check** scans the project for **known security vulnerabilities in third-party libraries** and generates a security report (dependency-check-report.xml).

**7. Trivy File System (FS) Scan**

This step scans the entire source code using **Trivy FS Scan**, checking for security vulnerabilities in files and dependencies. The results are stored in trivyfs.txt.

**8. Docker Build & Push**

* **Builds the Docker image** using the provided Dockerfile while passing the **TMDB API key** as a build argument.
* Tags the built image with the repository (karthik3513/netflix:latest).
* **Pushes the image** to the **DockerHub registry**.

**9. Trivy Image Scan**

After pushing the Docker image, **Trivy scans the built image for vulnerabilities**. The results are stored in trivyimage.txt for further analysis.

**10. Deploy to Container**

The **Netflix clone app** is deployed inside a **Docker container**, running on port **8081**, and is now accessible for testing.

A computer screen with text

Description automatically generated

**11. Post-Build Email Notifications**

An **email notification** is sent to the specified recipient, containing:  
**Build result (Success/Failure)**  
**Project name and build number**  
**Jenkins build URL**  
**Attached security reports from Trivy scans**

After writing the script, save the pipeline and build the job.

A screenshot of a computer

Description automatically generated

Meanwhile the job is being built, let us monitor the Jenkins metrics in Grafana.

A screenshot of a computer

Description automatically generated

As we can see here our job is executed successfully.

A screenshot of a computer

Description automatically generated

We have our Netflix docker image in our Docker hub.

A screenshot of a computer

Description automatically generated

**PHASE – 4**

**DEPLOYMENT**

**Deploy EKS Cluster**

A computer screen with blue text

Description automatically generated

A screenshot of a computer

Description automatically generated

Now install ArgoCD.

A screenshot of a computer

Description automatically generated

In the repositories section add the Git Repo.

A screenshot of a computer

Description automatically generated

Next we need to create an app in the ArgoCD which fetches the data or manifest files present in our GitHub Repository.

A screenshot of a computer

Description automatically generated

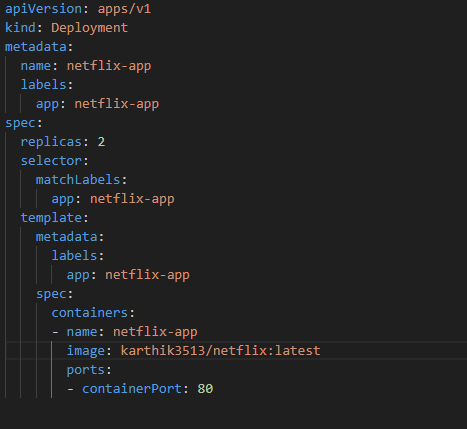
Mention the source as GitHub URL and path where the manifest files are present.

A screenshot of a chat

Description automatically generated

In the Kubernetes folder in GitHub, create 2 manifest files for deployment and service.

Deployment.yaml



This **Kubernetes Deployment** defines a **Netflix clone application** with **two replicas**, ensuring high availability and load balancing. The **matchLabels selector** (app: netflix-app) ensures that the pods are correctly managed under this deployment. Each pod runs a container using the **Docker image** karthik3513/netflix:latest, which contains the packaged application. The container listens on **port 80**, allowing external traffic routing when exposed via a **Kubernetes Service**. This setup ensures scalability, resilience, and seamless updates by automatically managing and restarting failed pods.

Service.yaml

A screen shot of a computer program

Description automatically generated

This **Kubernetes Service** exposes the **Netflix clone application** to external users by defining a **NodePort** service. The **selector (app: netflix-app)** ensures that traffic is routed to the pods managed by the corresponding **Deployment**. The service listens on **port 80** inside the cluster and forwards requests to the container's **target port 80**. The **nodePort 30007** allows external access through any node in the cluster on port **30007**, making the application reachable from outside the Kubernetes environment. This setup provides a stable networking layer, enabling seamless communication between users and the deployed app.

A screenshot of a chat

Description automatically generated

**Install Node\_exporter on both master and worker**

Let’s add Node\_exporter on Master and Worker to monitor the metrics

First, let’s create a system user for Node Exporter by running the following command:

sudo useradd \

--system \

--no-create-home \

--shell /bin/false node\_exporter

A black and white screen

Description automatically generated

At this point, we have only a single target in our Prometheus. There are many different service discovery mechanisms built into Prometheus. For example, Prometheus can dynamically discover targets in AWS, GCP, and other clouds based on the labels. In the following tutorials, I’ll give you a few examples of deploying Prometheus in a cloud-specific environment. For this tutorial, let’s keep it simple and keep adding static targets. Also, I have a lesson on how to deploy and manage Prometheus in the Kubernetes cluster.

To create a static target, you need to add job\_name with static\_configs. Go to Prometheus server

A screen shot of a computer code

Description automatically generated

Now synchronise the app in ArgoCD and we can see our application is hosted on one of the worker nodes on the port 30007.

A screenshot of a video game

Description automatically generated

Let us check the node metrics in the Grafana dashboard.

A screenshot of a computer

Description automatically generated

**SECURITY & COMPLIANCE BEST PRACTICES**

**Media Streaming Security**

* Content Protection: Preventing unauthorized access to video files and API endpoints.
* Access Controls: Implementing RBAC & JWT authentication for secure user access.

**Secure DevOps Pipeline**

* Vulnerability Scanning: Integrated SonarQube, Trivy, and OWASP ZAP scans.
* Immutable Deployments: Ensuring every deployment is containerized & versioned.
* Infrastructure Security: Kubernetes RBAC policies, network segmentation, and security policies.

**Monitoring & Incident Response**

* Real-time Monitoring: Detecting stream buffering issues, downtime, or latency spikes.
* Automated Incident Response: Integrated alerting for on-call rotation & escalation handling.

**BUSINESS IMPACT & KEY BENEFITS**

**Measurable Business Impact**

A screenshot of a computer

Description automatically generated

**Key Business & Operational Benefits**

* Seamless Streaming Experience – Ensuring zero buffering and optimized video delivery.
* Fast & Secure Deployments – Automated pipeline eliminates manual intervention delays.
* Scalable Infrastructure – Kubernetes auto-scaling ensures high traffic management.
* Optimized Cost & Resource Utilization – Efficiently manages compute resources.

**FUTURE ENHANCEMENTS & ROADMAP**

A screenshot of a computer

Description automatically generated

**CONCLUSION**

This **Netflix clone CI/CD pipeline** ensures a **secure, scalable, and highly available** media streaming experience. By leveraging **DevOps best practices, Kubernetes orchestration, and automated security scanning**, this implementation **enhances user experience, optimizes cloud resources, and accelerates software delivery**.

This **production-grade DevOps implementation** positions the Netflix clone as a **scalable, secure, and high-performance streaming platform**, enabling future innovations in the media industry.