**Amazon Prime Video Clone - CI/CD Pipeline**

* **Overview:**

This project focuses on implementing a Continuous Integration (CI) and Continuous Deployment (CD) pipeline using Jenkins. The goal is to deploy a clone of the Amazon Prime Video application to Amazon Elastic Container Registry (ECR) while incorporating security vulnerability scanning using Trivy. The pipeline also integrates SonarQube for static code analysis and quality gate checks.

* **Tools & Technologies:**
* **Jenkins**: An automation server used to manage the CI/CD pipeline.
* **SonarQube**: A platform for continuous inspection of code quality and security vulnerabilities.
* **Trivy**: A security vulnerability scanner specifically for Docker images.
* **Docker**: A containerization platform used to package the application into lightweight containers.
* **AWS ECR (Elastic Container Registry)**: A fully managed Docker container registry for storing Docker images.
* **AWS CLI**: Command-line tool to interact with AWS services.
* **Pipeline Stages:**

1. **Git Checkout**:  
   This stage pulls the source code from a GitHub repository, which contains the Amazon Prime Video clone project.
2. **SonarQube Analysis**:  
   Performs static code analysis using SonarQube to check for code smells, bugs, and security vulnerabilities. It uses the pre-configured SonarQube scanner.
3. **Quality Gate**:  
   Ensures that the code meets the defined quality standards using SonarQube’s Quality Gate. If the quality gate fails, the pipeline stops.
4. **Install npm Dependencies**:  
   This stage installs the required npm packages for the application using the command npm install.
5. **Trivy Security Scan**:  
   Scans the application’s directory for known vulnerabilities using Trivy. The scan results are stored in a file named trivy.txt.
6. **Build Docker Image**:  
   This stage builds the Docker image for the application using the Dockerfile found in the repository. The image is tagged based on the user-defined ECR repository name.
7. **Create ECR Repository (if it doesn’t exist)**:  
   The pipeline checks whether the specified ECR repository exists in the AWS account. If not, it creates the repository automatically. This step ensures that the Docker image has a valid destination for storage.
8. **Tag & Push Docker Image to AWS ECR**:  
   The Docker image is tagged with both a unique build number and the latest tag. Afterward, the image is pushed to the specified ECR repository in the user's AWS account.
9. **Delete Docker Images from Jenkins Server**

Once the images are pushed to ECR, the unused (dangling) docker images will be deleted to avoid storage issues.

1. **Create ArgoCD, Grafana & Prometheus**

The pipeline will create ArgoCD, Grafana & Prometheus. ArgoCD will be used to deploy Docker image from ECR. Grafana & Prometheus will be used for monitoring the application.

**INFRASTRUCTURE CONFIGURATION:**

* **EC2 Instance:**
* **Operating System**: Ubuntu 24.04
* **Security Group:**
* **SSH**: Port 22
* **HTTP**: Port 80
* **HTTPS**: Port 443
* **NodeJS**: Port 3000
* **Jenkins**: Port 8080
* **SonarQube**: Port 9000
* **Jenkins Plugins:**

1. **SonarQube Scanner**:  
   Used to integrate Jenkins with SonarQube to check code quality and perform quality gate checks.
2. **NodeJS**:  
   Manages and allows the use of different versions of Node.js within Jenkins.
3. **Pipeline: Stage View**:  
   Provides a visual interface to monitor the stages of the Jenkins pipeline.
4. **Docker**:  
   Allows Jenkins to manage Docker containers directly.
5. **Docker Commons**:  
   Facilitates the sharing of common Docker configurations across Jenkins jobs.
6. **Docker Pipeline**:  
   Supports building and running Docker containers within Jenkins pipelines.
7. **Docker API**:  
   Provides an API to interact with Docker within Jenkins.
8. **docker-build-step**:  
   Adds Docker-related build steps, like building images or running containers in freestyle jobs.
9. **SSH Agent**:  
   Manages SSH credentials for remote server access within Jenkins pipelines.
10. **Eclipse Temurin Installer**:  
    Used to install OpenJDK within Jenkins.
11. **Prometheus Metrics**:  
    Enables Jenkins to expose metrics for monitoring and alerting.

**SONARQUBE CONFIGURATION:**

* **SonarQube Webhook:**

The SonarQube webhook is used to send analysis results back to Jenkins.

* **Configuration**:
  + Name: sonarqube-webhook
  + URL: http://<jenkins-url>:8080/sonarqube-webhook/
  + Secret: None
* **SonarQube Token:**
* Generated token to authenticate Jenkins with SonarQube for analysis and quality gate checks.
* **Credentials:**
* **SonarQube Token:** Used to authenticate and interact Jenkins with SonarQube.
* **AWS credentials (Access Key and Secret Key):** These credentials are used to authenticate and interact Jenkins with AWS, to push the Docker Image to AWS ECR.
* **Jenkins SonarQube System Configuration:**
* Helps to store SonarQube server details globally in Jenkins instead of specifying them in each pipeline script.
* **Configuration**:
  + Name: sonar-server
  + URL: http://<sonar-server-url>:9000

**JENKINS GLOBAL TOOL CONFIGURATION:**

* **JDK Installation:**
* **Name**: jdk17
* **Install From**: adoptium.net
* **Version**: jdk17.0.8.1+1
* **SonarQube Scanner Installation:**
* **Name**: sonar-scanner
* **Install From**: Maven Central
* **Version**: 6.2.0.4584
* **NodeJS Installation:**
* **Name**: node16
* **Install From**: nodejs.org
* **Version**: 16.20.0
* **Docker Installation:**
* **Name**: docker
* **Install From**: docker.com
* **Version**: Latest
* **Flow of Execution:**

1. **Source Code Fetching**: The pipeline begins by pulling the code from GitHub.
2. **Static Analysis**: SonarQube performs code quality and security checks on the pulled code.
3. **Build & Testing**: The application’s dependencies are installed, and the application is packaged into a Docker image.
4. **Security Scanning**: Trivy scans the Docker image for vulnerabilities and produces a report.
5. **Deployment**: The final Docker image is pushed to AWS ECR, making it ready for deployment to AWS ECS or EKS. The application is deployed using ArgoCD. The pods are monitored using Grafana & Prometheus.

* **Conclusion:**

This project provides a comprehensive CI/CD pipeline from source code management to deployment with integrated code quality checks and security scans. Using AWS ECR ensures that the Dockerized application is securely stored and readily available for deployment in a scalable AWS environment. The application is deployed using ArgoCD. The pods are monitored using Grafana & Prometheus.