https://www.youtube.com/watch?v=pbGA-B\_SCVk

Step 1 — Launch an Ubuntu(22.04) T2 Large Instance

Step 2 — Install Jenkins, Docker and Trivy. Create a Sonarqube Container using Docker.

Step 3 — Create a TMDB API Key.

Step 4 — Install Prometheus and Grafana On the new Server.

Step 5 — Install the Prometheus Plugin and Integrate it with the Prometheus server.

Step 6 — Email Integration With Jenkins and Plugin setup.

Step 7 — Install Plugins like JDK, Sonarqube Scanner, Nodejs, and OWASP Dependency Check.

Step 8 — Create a Pipeline Project in Jenkins using a Declarative Pipeline

Step 9 — Install OWASP Dependency Check Plugins

Step 10 — Docker Image Build and Push

Step 11 — Deploy the image using Docker

Step 12 — Kubernetes master and slave setup on Ubuntu (20.04)

Step 13 — Access the Netflix app on the Browser.

Building a Container Application Deployment Pipeline to Amazon ECR - Capstone Project

|  |  |  |  |
| --- | --- | --- | --- |
| **Deliverables** |  |  |  |
| A fully functional CI/CD pipeline for container applications. |  |  |  |
| Automation of image building and tagging. |  |  |  |
| Documentation of the pipeline setup and deployment process. |  |  |  |
| Testing and verification report. |  |  |  |
|  |  |  |  |
| **Tasks/Activities List** |  |  |  |
| **Research and Define Requirements:** |  |  |  |
| Identify the specific requirements for your containerized application. |  |  |  |
| Determine the technology stack, programming languages, and dependencies. |  |  |  |
| **Design Container Architecture:** |  |  |  |
| Decide on the containerization technology (e.g., Docker). |  |  |  |
| Create a Dockerfile that defines the container's environment and dependencies. |  |  |  |
| Set up a version control system (e.g., Git) to manage your application code. |  |  |  |
| **Setup Amazon ECR Repository:** |  |  |  |
| Create one or more Amazon ECR repositories to store your container images. |  |  |  |
| Set permissions and access policies for ECR repositories, following AWS IAM best practices. |  |  |  |
| **Establish AWS Infrastructure:** |  |  |  |
| Create an AWS CodePipeline to manage the CI/CD workflow. |  |  |  |
| Configure AWS CodeBuild as the build environment for your container image. |  |  |  |
| Set up an Amazon S3 bucket to store build artifacts and deployment files. |  |  |  |
| **Implement CI/CD Pipeline:** |  |  |  |
| Create a CodePipeline that includes source code integration from your version control system. |  |  |  |
| Configure build triggers to automatically initiate the pipeline when code changes occur. |  |  |  |
| Define build and deployment stages within the pipeline, specifying dependencies between stages. |  |  |  |
| **Automate Image Building:** |  |  |  |
| Configure AWS CodeBuild with build instructions, typically defined in a buildspec.yml file. |  |  |  |
| Use the build environment to retrieve source code, run tests, and build the Docker image. |  |  |  |
| Automatically tag the Docker image with a version or unique identifier. |  |  |  |
| **Security and Best Practices:** |  |  |  |
| Implement security measures such as scanning container images for vulnerabilities using tools like AWS ECR Image Scanning or third-party solutions. |  |  |  |
| Utilize IAM roles to manage access permissions for the pipeline and ECR repositories. |  |  |  |
| Implement encryption for container image storage in ECR. |  |  |  |
| **Testing and Validation:** |  |  |  |
| Develop unit and integration tests to ensure the application functions correctly within the container. |  |  |  |
| Implement automated testing within the CI/CD pipeline to catch potential issues early. |  |  |  |
| Conduct thorough testing of the entire pipeline to validate its functionality. |  |  |  |
| **Optimization:** |  |  |  |
| Review the pipeline for opportunities to optimize performance, resource utilization, and cost-efficiency. |  |  |  |
| Consider implementing caching mechanisms or parallelization to speed up the build process. |  |  |  |
| **Documentation and Reporting:** |  |  |  |
| Create comprehensive documentation for the CI/CD pipeline setup, including pipeline architecture, IAM roles, and Dockerfile structure. |  |  |  |
| Generate a report summarizing the pipeline's success, performance metrics, and any issues encountered and resolved. |  |  |  |
| **Monitoring and Logging:** |  |  |  |
| Set up monitoring and alerting using AWS CloudWatch to track pipeline performance and detect anomalies. |  |  |  |
| Configure logging to capture build and deployment logs for auditing and troubleshooting. |  |  |  |
| **Success Metrics** |  |  |  |
| A functional CI/CD pipeline for container application deployment. |  |  |  |
| Automated image building, tagging, and pushing to Amazon ECR. |  |  |  |
| Security measures applied to container image management. |  |  |  |
| Clear and comprehensive documentation of the pipeline setup. |  |  |  |
| **Bonus Points** |  |  |  |
| Implement deployment strategies (e.g., blue/green) in the CI/CD pipeline. |  |  |  |
| Integrate AWS CloudWatch for monitoring and logging. |  |  |  |
| Explore container orchestration tools like Amazon ECS or Kubernetes. |  |  |  |
| Implement advanced IAM policies for security and permissions |  |  |  |