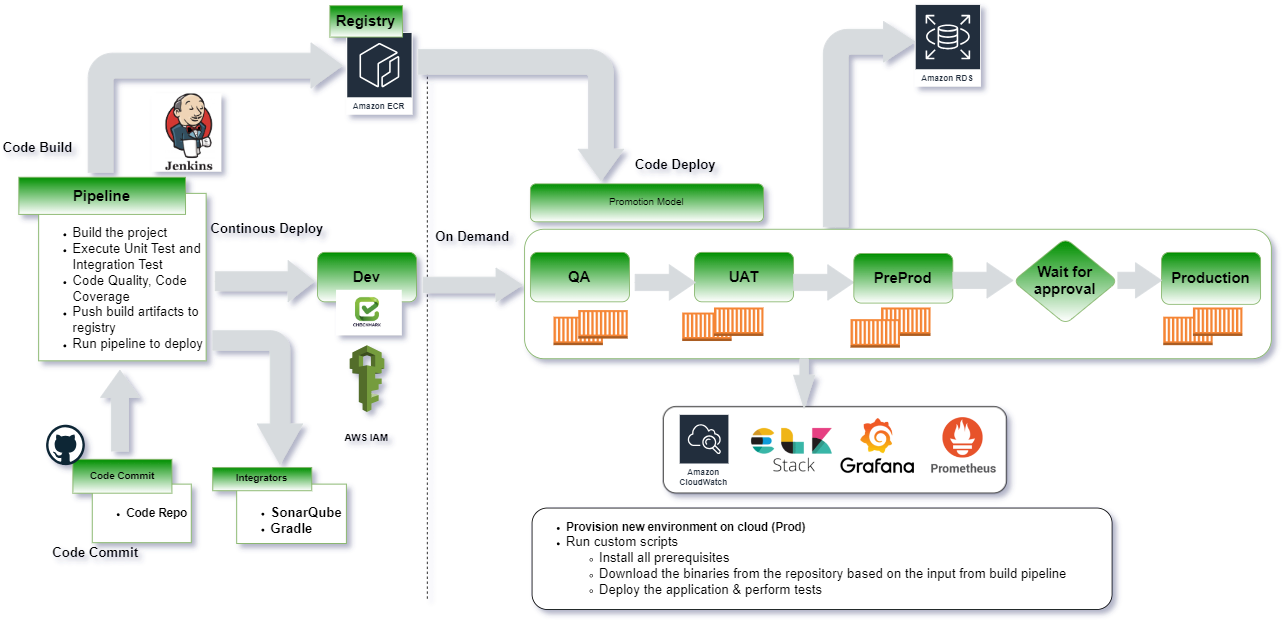
## Proposed DevOps Approach

The proposed deployment approach is integrated with the existing FreightVerify System/platform.  Continuous Integration and Continuous Deployment are configured with the existing tools. The services are isolated and deployed in the specific namespace in the AWS account.

The below table gives the details about the tool's integration.

|  |  |
| --- | --- |
| **Tools** | **Description** |
| Elastic Kubernetes Service | EKS-managed cluster is distributed across the region with High Availability, Autoscaling will be configured as part of the replica. Ingress Controller (AWS ALB) will be configured for path-based routing. |
| Elastic Container Registry | Docker images are created with JDK base image to run the jar files for the containers to be deployed on EKS. We use ECR centralized storage for configuring and uploading the docker images of non-prod and prod environments. |
| Terraform | The infrastructure will be defined as code using Terraform, enabling consistent and repeatable provisioning of AWS (Amazon Web Services) resources. This approach ensures the entire infrastructure is version-controlled, allows for easy tracking of changes, and simplifies collaboration. |
| Cloud Watch | This is centralized and configured in the region for extracting and maintaining the logs. |
| CloudFront | CloudFront signed URLs provide a mechanism to control access to the content served through a distribution. |
| AWS S3 | S3 bucket can be used to host the static website and to store the application-processed Documents, images, and reports. |
| AWS API Gateway | Configuring API Gateway to manage API endpoints. It helps for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. |
| SES | Simple Email Service (SES) is a flexible, highly scalable, and cost-effective service that allows application developers to send emails from within the application. We can use it for transactional emails, marketing emails, and sending emails in bulk. |
| SQS | Simple Queue Service (SQS) is a fully managed service offering from Amazon. It helps to decouple and scale microservices, serverless applications, and distributed systems. It removes the complexity and overhead associated with managing and operating message-oriented middleware. |
| SNS | Simple Notification Service (SNS) is a fully managed service offered by Amazon. It is helpful for application-to-application (A2A) and application-to-person (A2P) communication. Its A2A functionality offers high-throughput, push-based, many-to-many messaging between microservices, event-driven serverless applications, and distributed systems. |
| AWS RDS | It enables you to set up, operate, and scale a relational database in the cloud without needing to manage the underlying infrastructure.  With AWS RDS, there are options to choose from various database engines, including Amazon Aurora, MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server. RDS takes care of routine database administration tasks such as backups, software patching, and automatic scaling. |
| AWS Secret Manager | AWS Secrets Manager is a service provided by Amazon Web Services (AWS) that helps to protect sensitive information such as API keys, database credentials, and other secrets. It enables you to securely store, manage, and retrieve secrets for your applications and services. |
| AWS Lambda | AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). It allows you to run your code without provisioning or managing servers, enabling you to focus on writing code and building applications without the need to worry about infrastructure. |

CI/CD Approach:



The following are the steps execution as part of the pipeline the CICD pipeline is configured with all the requirement stages for the build execution:

* CICD Pipeline is configured with a multi-stage build process for the pipeline build execution.
* Continuous Integration of the build process starts and triggers the staged build process.
* The triggered build process on the compilation is successful proceed with the next gated check with the code quality.
* The pipeline job triggers to deploy the REST Microservices to Lambda functions,
* During the process, the image is validated for all the security integration with the scanning using the tool.
* The image of having the vulnerability criteria falls into criteria like high, medium, and low grouping.
* The rules are being set with the expectation for the code to be passed in and uploaded onto the Registry.
* On successful compilation the build is uploaded onto the Container Registry.
* The artifact is then called using the Continuous Integration process for the artifact deployment onto the K8 cluster.

**Release process**

* The releases are done using the templates configured with the chart for the specific repos and the same can be reused across the board. The releases are version-controlled, and the rollback can be done at any point of time.
* All the artifacts are stored on the Container Repo and tagged with the defined labels.
* The promotion model is being set so that the same artifact can be promoted onto the next environment using the pipeline.
* Non-prod environment setup done for doing their testing without disturbing any existing environment.
* Alerts are being configured at all levels in case of failure to trigger so that they are completely monitored with the tools.

For continuous integration and continuous deployment (CI/CD), we will utilize the following tools and practices:

* **Code repository**: GitHub and GitLab for version control and collaboration.
* **Code Quality**: SonarQube (formerly Sonar) is an open-source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs.
* **Build and dependency management**: Maven for building the Java-based microservices and React JS applications.
* **Containerization**: Docker will be used to containerize the microservices.
* **Container Registry**: AWS Elastic Container Registry (ECR) will be used to store and manage Docker images.
* **Continuous Integration and Deployment**: Jenkins will be used to establish a robust CI/CD pipeline. It will integrate with the version control system (e.g., Git) to trigger automated builds, perform code quality checks, and deploy the applications to the EKS cluster. Jenkins pipelines will be defined using declarative syntax for easy maintenance and scalability.
* **Infrastructure Updates**: Infrastructure updates will be handled using Terraform, triggered by the Jenkins CI/CD pipeline.

Application and Infrastructure Monitoring

To monitor the deployed infrastructure and applications, we will implement the following:

* **Prometheus and Grafana**: Prometheus and Grafana are widely used open-source tools for monitoring, logging, and alerting in Kubernetes environments, including AWS EKS.

Prometheus will send the metrics data it collects from the AWS EKS Cluster, and Grafana uses it to create the visualizations.

* **Application Logging**: We can integrate Grafana Loki, a log aggregation and search system, with your EKS cluster. Grafana Loki helps you collect and store logs from various sources within your EKS cluster, including application logs, container logs, and system logs.

Configure Grafana Loki to collect and index logs and use Grafana to search, filter, and visualize log data.

* **Alerting with Prometheus Alert Manager:** Configure Prometheus Alert Manager to receive and manage alerts generated by Prometheus. Define alerting rules in Prometheus based on your monitoring requirements. These rules define the conditions that trigger alerts. Configure Alert Manager to send alerts via various notification channels such as email, Slack, or other custom integrations.

Branching strategy

A diagram of a software project

Description automatically generated

**Proposed Flow**:

* Standard conventions are followed for the branching strategy so that the product is uniquely identified.
* The feature branches are short-lived branches that are being developed and merged into the development branches.
* The development branches are then merged into the supporting branches called the Release branches.
* Tags are being created from the branches to do the releases.
* The release is then tagged based on the short outcomes of the releases.
* Finally approved changes are merged back into the Master branch and the deployment happens from this branch.
* The master branch is then again branched out for the latest changes in development/feature to be worked upon by developers.

The hotfix branches are created in case of any critical issues to be fixed on the production release being done from the master branch.

Release Process

* The release process will be created with the managers & cab approval obtained prior to the releases.
* Release processes are defined with the CI/CD Integrated
* Deployment will be done through the build tool.
* Frequent branches and the tagging process will be created for the deployment.
* Release will be tagged and stored as a Tag in the Repo.
* Hotfix branches will be created during the process of any critical fixes that need to be part of the deployment.
* Communicate to the stakeholders on the release complete
* The Release will be deployed using the pipeline with the specific branching Tag created for the deployment.
* The Releases will be version-controlled and deployed from the specific pipeline.
* The promotion of artifacts will be done using pipelines.

Blue-Green and Canary Deployment Integration:

We are aimed at integrating Blue-Green and Canary Deployment strategies into your software release process. By implementing these strategies, we aim to minimize downtime, reduce risks, and enhance the overall reliability and quality of your software deployments.

**Blue-Green Deployment Strategy:**

We recommend setting up a Blue-Green deployment model, where we maintain two identical environments: the "Blue" environment represents the current stable version, while the "Green" environment represents the new version or feature being deployed. The key steps of the proposed integration are as follows:

* Develop an automated deployment pipeline using the existing tool Jenkins to facilitate the deployment process.
* Implement automated testing suites that run against both the Blue and Green environments to ensure functional and performance validation.
* Define release criteria and establish a rollback strategy to quickly revert to the stable version (Blue environment) in case of any issues or unexpected behavior during the deployment.
* Gradually switch traffic from the Blue environment to the Green environment using load balancers or DNS routing, minimizing downtime and allowing for real-time monitoring and validation.
* Monitor key performance indicators (KPIs) and track user feedback to ensure a successful deployment and identify any potential issues that may require further attention.

**Canary Deployment Strategy:**In addition to Blue-Green deployments, we recommend incorporating a Canary deployment strategy to gradually roll out changes to a subset of users or systems before a full release. This will allow for early detection of any issues or performance problems. The proposed integration steps include:

* Identify a subset of users or systems that will act as the Canary group for each release.
* Implement feature flagging or traffic routing mechanisms to direct a portion of traffic to the Canary group while the majority of users remain on the stable version.
* Monitor key metrics and performance indicators for the Canary group, comparing them against predefined thresholds and service level objectives (SLOs).
* Gradually increase the proportion of traffic served to the Canary group as the release progresses, closely monitoring the impact on metrics and user experience.
* Based on the performance and feedback received, decide whether to proceed with a full release or roll back the changes.

**Benefits and Deliverables:**

* Minimized downtime and reduced risks during software deployments, ensuring a seamless user experience.
* Enhanced reliability and quality of deployments through automated testing and validation.
* Real-time monitoring and metrics tracking to measure the success and impact of each deployment.
* Improved feedback loop and user experience by gradually rolling out changes to a subset of users before full release.
* Documentation and knowledge sharing on best practices for Blue-Green and Canary deployments.

DevOps Cloud Security for Application

* Security Virtual Private Cloud will be configured.
* Security Rules for the private and public cloud configured.
* Access Identity and Management (IAM) Access Analyzer
* Key Management Service Integrated for encrypting the keys used by the applications and supported AWS services in multiple Regions around the world from a single console.
* S3 configuration and monitoring
* Security assessments on the firewall rules & configuration
* Access to the corresponding repos

Cost-Effectiveness, High Availability, and Scalability:

To achieve cost-effectiveness, high availability, and scalability, we will employ the following strategies:

* **AWS Auto Scaling**: EKS clusters can be configured with AWS Auto Scaling, allowing the cluster to scale horizontally based on demand and workload metrics.
* **Load Balancing**: We will utilize AWS Elastic Load Balancer (ELB) or AWS Application Load Balancer (ALB) to distribute traffic across multiple instances of the microservices and React JS application, ensuring high availability and load distribution.
* **Cost Analysis:** Cost analysis will be performed to optimize expenses while maintaining a highly available and scalable architecture. This includes right-sizing EC2 instances, using spot instances for non-critical workloads, leveraging reserved instances for cost savings, and utilizing auto-scaling features to match resource usage to demand.