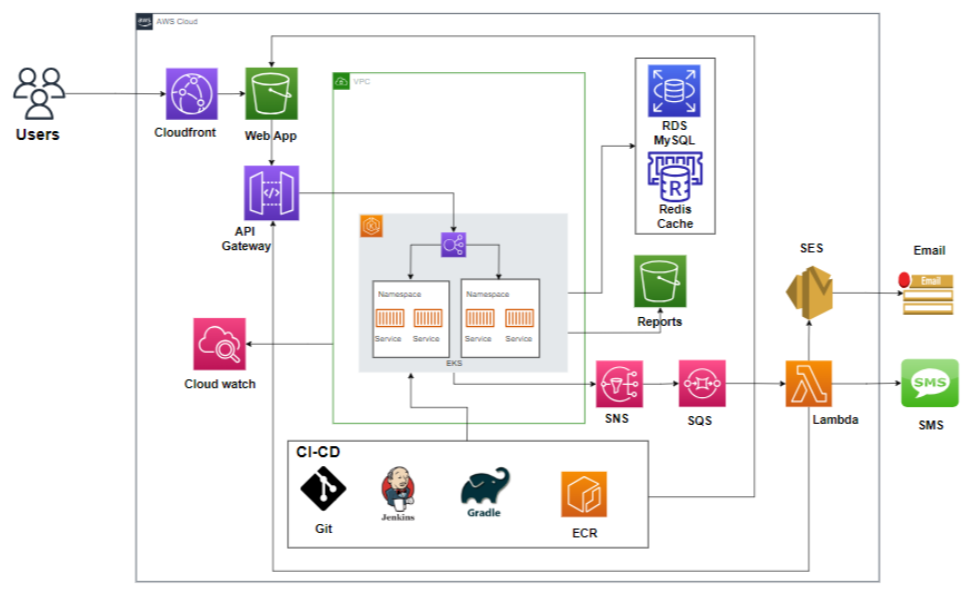
## Proposed DevOps Approach - Yogeesh

## Deployment Architecture

The proposed deployment architecture is integrated with the Kubernetes (AWS Elastic Kubernetes service) platform.  The Continuous Integration and Continuous Deployment being configured either with the proposed tools or with the existing tools. The services are isolated and deployed in the specific namespace in AWS account.

The diagram below and the tables 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. |
| Elastic Container Registry | Base images are created for the containers to be deployed on EKS. We need centralized storage for configuring and uploading the 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. |
| Cloud Trail | Enables the auditing and monitoring of API calls made with the AWS Accounts. |
| AWS Elastic Cache for Redis | Create cross-Region read replica clusters for ElastiCache for Redis to enable low-latency reads.  Disaster recovery across AWS Regions. |
| 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 ReactJS static web application and to store the processed Documents 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. |
| AWS Lambda | Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services. This can be used to deploy your APIs. |
| 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. |

## 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.
* 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 on to 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 in 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 is done for doing their testing without disturbing any existing environment.
* Performance test scripts are integrated to run on the code deployed and validated.
* 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 for version control and collaboration.

**Build and dependency management**: Gradle for building the Spring Boot microservices and React JS application.

**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, execute tests, 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.

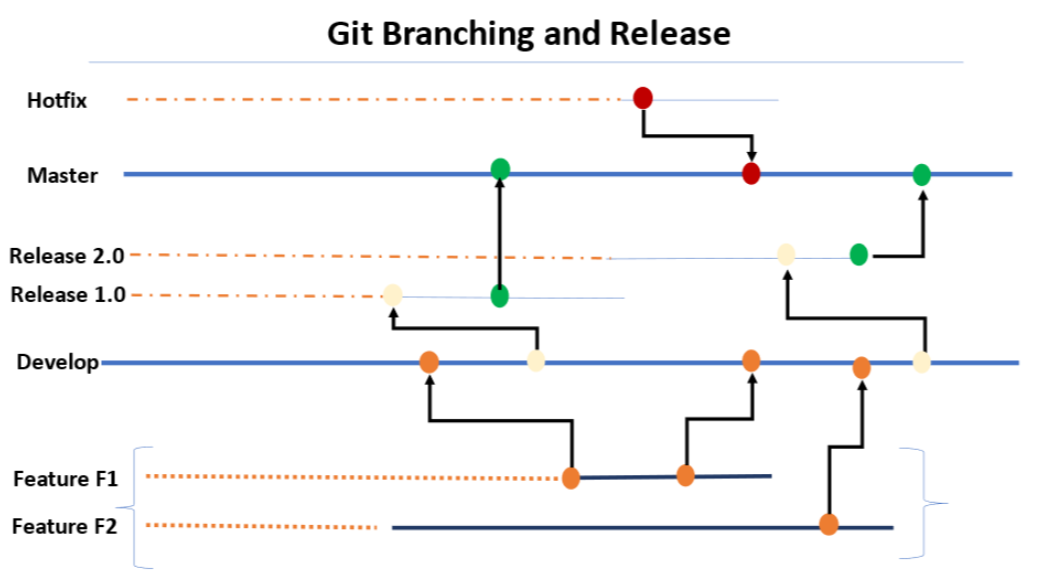
Prometheus will send the metrics data it collects from the AWS EKS Cluster, and Grafana uses it to create the visualizations.

* **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 any other custom integrations.

## Branching strategy



Proposed Flow:

* Standard conventions are followed for the branching strategy so that the product is uniquely identified.
* The feature branches are short lived branches which are being developed and merged into the development branches.
* The development branches are then merged on to 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 being 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 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 Stake holders 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.

## 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 encryption.
* S3 configuration and monitoring
* Security assessments on the firewall rules & configuration
* Access to the corresponding repos

## Disaster Recovery:

To ensure high availability and disaster recovery, we will implement the following measures:

* **EKS Multi-AZ Deployment**: The EKS cluster will be deployed across multiple Availability Zones (AZs) to achieve redundancy and fault tolerance.
* **Automated Database Backups**: AWS RDS with MySQL supports automated backups, allowing us to easily restore data in case of a failure.
* **Cross-Region Replication**: For critical workloads, we can set up cross-region replication of the RDS database to a different AWS region, ensuring data redundancy and disaster recovery.

## 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.